First Hit Fwd Refs
**End of Result Set**

Generate Collection Print

L7: Entry 1 of 1 File: USPT May 20, 2003

US-PAT-NO: 6567813
DOCUMENT-IDENTIFIER: US 6567813 B1

TITLE: Quality of service maintenance for distributed collaborative computing

DATE-ISSUED: May 20, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Zhu; Min | Los Altos Hills | CA | | |
| Zhao; Bin | San Jose | CA | | |

US-CL-CURRENT: 707/100

CLAIMS:

We claim:

1. A method of distributed collaborative computing comprising: partitioning a collaboration function into sub-functions; assigning at least one said sub-function to each of a plurality of logical processes; associating a respective management process with each of said plurality of logical processes, said logical processes configured so that each said logical process is capable of communicating with every other said logical process thru said respective management process; communicating between said logical processes using said respective management processes; and monitoring said respective management processes with a single supervisor process; wherein said monitoring further comprises: obtaining state information; combining said state information with load information to create a quality of service (QoS) value; and modifying said partitioning based on said QoS value.

2. The method of claim 1, wherein said modifying said partitioning comprises optionally load shedding.

3. The method of claim 1, wherein said modifying said partitioning comprises optionally spawning additional logical processes.

4. A computer program for use in distributed collaborative computing, comprising computer instructions for: partitioning a collaboration function into sub-functions; assigning at least one said sub-function to each of a plurality of logical processes; associating a respective management process with each of said plurality of logical processes, said logical processes configured so that each said logical process is capable of communicating with every other said logical process thru said respective management process; communicating between said logical processes using said respective management processes; and monitoring said respective management processes with a single supervisor process; wherein said monitoring further comprises: obtaining state information;

h e b b g eeef c e c e ge

combining said state information with load information to create a quality of service (QoS) value; and modifying said partitioning based on said QoS value.

5. The computer program of claim 4, wherein said modifying said partitioning comprises optionally load shedding.

6. The computer program of claim 4, wherein said modifying said partitioning comprises optionally spawning additional logical processes.

7. A computer-readable medium storing a computer program executable by a plurality of server computers, the computer program comprising computer instructions for: partitioning a collaboration function into sub-functions; assigning at least one said sub-function to each of a plurality of logical processes; associating a respective management process with each of said plurality of logical processes, said logical processes configured so that each said logical process is capable of communicating with every other said logical process thru said respective management process; communicating between said logical processes using said respective management processes; and monitoring said respective management processes with a single supervisor process; wherein said monitoring further comprises: obtaining state information; combining said state information with load information to create a quality of service (QoS) value; and modifying said partitioning based on said QoS value.

8. The computer-readable medium of claim 7, wherein said modifying said partitioning comprises optionally load shedding.

9. The computer-readable medium of claim 7, wherein said modifying said partitioning comprises optionally spawning additional logical processes.

10. A computer data signal embodied in a carrier wave, comprising computer instructions for: partitioning a collaboration function into sub-functions; assigning at least one said sub-function to each of a plurality of logical processes; associating a respective management process with each of said plurality of logical processes, said logical processes configured so that each said logical process is capable of communicating with every other said logical process thru said respective management process; communicating between said logical processes using said respective management processes; and monitoring said respective management processes with a single supervisor process; wherein said monitoring further comprises: obtaining state information; combining said state information with load information to create a quality of service (QoS) value; and modifying said partitioning based on said QoS value.

11. The computer data signal of claim 10, wherein said modifying said partitioning comprises optionally load shedding.

12. The computer data signal of claim 10, wherein said modifying said partitioning comprises optionally spawning additional logical processes.

# Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

**Search Results** - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 6567813 B1

L7: Entry 1 of 1 File: USPT May 20, 2003

US-PAT-NO: 6567813
DOCUMENT-IDENTIFIER: US 6567813 B1

TITLE: Quality of service maintenance for distributed collaborative computing

DATE-ISSUED: May 20, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Zhu; Min | Los Altos Hills | CA | | |
| Zhao; Bin | San Jose | CA | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| WebEx Communications, Inc. | San Jose | CA | | | 02 |

APPL-NO: 09/ 752377 [PALM]
DATE FILED: December 29, 2000

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/100
US-CL-CURRENT: 707/100

FIELD-OF-SEARCH: 707/101, 707/102, 707/104.1, 707/100

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

| PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|
| 4654484 | March 1987 | Reiffel et al. | 379/53 |
| 5434852 | July 1995 | LaPorta et al. | 370/58.2 |
| 5577188 | November 1996 | Zhu | 395/326 |
| 5623603 | April 1997 | Jiang et al. | 395/200.04 |
| 5764901 | June 1998 | Skarbo et al. | 395/200.34 |
| 5805804 | September 1998 | Laursen et al. | 395/200.02 |

h e b b g e e e f e c ef b e

| | | | |
|---|---|---|---|
| 5829001 | October 1998 | Li et al. | 707/10 |
| 5841980 | November 1998 | Waters et al. | 709/204 |
| 5887170 | March 1999 | Ansberry et al. | 395/687 |
| 5892946 | April 1999 | Woster et al. | 395/680 |
| 5916302 | June 1999 | Dunn et al. | 709/204 |
| 5944791 | August 1999 | Scherpbier | 709/218 |
| 5974446 | October 1999 | Sonnenreich et al. | 709/204 |
| 5996002 | November 1999 | Katsurabayashi et al. | 709/204 |
| 6038593 | March 2000 | Huckins | 709/217 |
| 6055574 | April 2000 | Smorodinsky et al. | 709/226 |
| 6058416 | May 2000 | Mukherjee et al. | 709/203 |
| 6167432 | December 2000 | Jiang | 709/204 |
| 6182085 | January 2001 | Eichstaedt et al. | 707/104.1 |
| 6246444 | June 2001 | Kim | 348/614 |
| 6263433 | July 2001 | Robinson et al. | 713/100 |
| 6289385 | September 2001 | Whipple et al. | 709/229 |
| 6397191 | May 2002 | Notani et al. | 705/9 |

OTHER PUBLICATIONS

Boyd, Lane, "Taking Collaboration Into Orbit," Computer Graphics World, Sep., 1998.
Caton, Michael, PC Week, "Easy Access, Low Cost Make Collaboration A Good Outsourced Fit," Feb. 28, 2000.
Downs, Scott and Lange, Alex, Open Systems Today, "The Net As A Vehicle For Business Data Sharing," Apr. 25, 1994.
Gall, Ulrich; Hauck, Franz J., "Promondia: A Java-Based Framework For Real-Time Group Communication In the Web," Apr. 7-11, 1997, Internet Publication.
Ly, Eric, "Distributed Java Applets For Project Management On The Web," May/Jun., 1997, IEEE Internet Computing Online, vol. 1, No. 3, Internet Publication.
Mates, Nathan; Nystrom, Mika; Schooler, Eve, "The Web Meets MOOs, IRC And the MBone," Jun. 8, 1995, Internet Publication.
Meyer, Tom; Blair, David; Hader, Suzanne, "A MOO-Based Collaborative Hypermedia System for WWW," Internet Publication.
Novitski, B.J., Computer Graphics World, "Team Building," Apr. 1, 2000.
Roberts-Witt, Sarah L., Internet World, "Online Collaboration Tools Maker Targets A Maturing Market," Jun. 15, 2000.
Selvaratnam, Nirooban, "Overview Of Programming Languages And Support Tools For Distributed Programming," Internet Publication.

ART-UNIT: 2171

PRIMARY-EXAMINER: Metjahic; Safet

ASSISTANT-EXAMINER: Al-Hashemi; Sana

ATTY-AGENT-FIRM: Austin; Sidley Brown & Wood LLP Woo; Philip W.

ABSTRACT:

A distributed collaborative computer system is provided that comprises a plurality of server computers interconnected via a high-speed link. Client computers can connect to any available server computer and start or join a conference hosted on either the server computer to which the client computer is connected or any other server in the system. As a result, the system and method of the present invention

is easily scalable to support an arbitrary number of participants to a conference by merely adding the appropriate number of server computers to the system. In addition, by replicating the conference information on more than one server computer, the single point of failure limitation is eliminated. In fact, if a server hosting or participating in a conference malfunctions, the failure is detected by other server computers and the client computer is able to reconnect to the conference through a new server computer.

12 Claims, 34 Drawing figures



| Term | Documents |
|---|---|
| (5 AND 6).USPT. | 1 |
| (L5 AND L6).USPT. | 1 |



h e b b g e e e f e c ef b e

L53: Entry 3 of 12 File: USPT Jun 24, 2003

DOCUMENT-IDENTIFIER: US 6584466 B1
TITLE: Internet document management system and methods

Abstract Text (1):
An Internet-based document management system and methods are provided wherein an electronic document may be stored on an Internet-accessible server and accessed using a previously known web browser, downloaded for review or manipulation, and then returned to the server for access by further users. The server is programmed to provide a plurality of services supported by a common database and document store, including storage and retrieval services, an electronic document delivery service, a document distribution service, a collaborative file sharing service and a workflow service. The system preferably also is programmed with a security function, a filtering function, accounting functions that enable detailed accounting of transactions occurring on the system, and a customization function that permits multiple service providers to utilize the common document management services of a server, while presenting end-users with distinct dedicated websites.

Brief Summary Text (4):
Document management systems are known that permit multiple users to store and retrieve electronic documents on a closed client/server architecture network, such as a local area network or wide area network. These previously known document management systems, such as DOCSFusion, available from PCDOCS, Inc., Toronto, Ontario, Canada and EDMS 98, available from Documentum, Inc., Pleasanton, Calif., require the presence of a client application on each node of the network that is to access and manipulate files.

Brief Summary Text (6):
Smith U.S. Pat. No. 5,790,790 describes an Internet electronic document delivery system, wherein an e-mail message contains a direct reference (i.e., a Uniform Resource Locator or "URL") to an electronic document stored on a server. When a recipient receives the e-mail message, the direct reference is used to access the document. A drawback of the system described in that patent, is that the sending computer must include a specialized client application for interacting with the server. The system described in that patent also lacks the kinds of transaction logging and accounting functions needed to provide a useful document management system.

Brief Summary Text (7):
The POSTA.RTM. system, offered by Tumbleweed Software Corporation, Redwood City, Calif., overcomes some of the drawbacks of the system described in the foregoing patent. For example, the POSTA.RTM. system eliminates the need for specialized client software for basic document delivery operations, and permits the use of a previously known web browser, such as Internet Explorer 4.0.RTM., available from Microsoft Corp., Redmond, Wash., or Netscape Navigator.RTM., Netscape Corporation, Mountain View, Calif. That commercial system also eliminates use of the direct reference in the e-mail message, instead providing a URL for a webpage that provides the user with several options for document delivery. The system provides none of the capabilities normally associated with a document management system.

Brief Summary Text (8):

Higley U.S. Pat. No. 5,790,793, like the foregoing Smith patent, also describes an Internet electronic document delivery system wherein an e-mail message includes a URL reference to a document stored in a server. This system described in this patent also requires the use of a specialized client application, and is limited to an electronic document delivery service.

Brief Summary Text (10):
In view of the foregoing it would be desirable to provide a document management system and methods that permit electronic documents to be made available for use on open systems, such as the Internet, and to be accessed using a previously known web browser--without the need for a specialized client application.

Brief Summary Text (14):
It yet further would be desirable to provide an Internet-based document management system and methods that enable the transaction logging and accounting functions needed for multi-user collaborative electronic document manipulation, for example, so that revisions to a document may be tracked.

Brief Summary Text (15):
It also would be desirable to provide an Internet-based document management system and methods that enable tracking of transactions performed on a document for billing purposes, and which provide needed access-control protocols, for example, so that specific users' privileges with respect to a document may be defined.

Brief Summary Text (17):
In view of the foregoing, it is an object of this invention to provide a document management system and methods that permit electronic documents to be made available for use on open systems, such as the Internet, and to be accessed using previously known web browser--without the need for a specialized client application.

Brief Summary Text (21):
It is a further object of this invention to provide an Internet-based document management system and methods that enable the transaction logging and accounting capabilities needed for multi-user collaborative electronic document manipulation, for example, so that revisions to a document may be tracked.

Brief Summary Text (22):
It is a still further object of the present invention to provide an Internet-based document management system and methods that enable tracking of transactions performed on a document for billing purposes, and which provide needed access-control protocols, for example, so that specific users' privileges with respect to a document may be defined.

Brief Summary Text (23):
These and other objects of the present invention are accomplished by providing an Internet-based document management system and methods wherein an electronic document may be stored on an Internet-accessible server and accessed using a previously known web browser, downloaded for review or manipulation, and then returned to the server for access by further users. In.accordance with the principles of the present invention, the server is programmed with several routines that perform numerous functions, referred to hereinafter as "services," that provide a full-featured document management system.

Brief Summary Text (24):
In a preferred embodiment, the document management system is programmed to provide a plurality of services supported by a common database and document store. These services preferably include storage and retrieval services to and from an Internet-based storage site, an electronic document delivery service, a collaborative file sharing service and a workflow service, and a document distribution service. The server also preferably is programmed to perform a security function, to verify or

define a requestor's ability to access an electronic document, a filtering function that performs selective or automatic filtering of documents during storage to and/or retrieval from the storage site, and accounting functions that enable detailed accounting of, for example, usage of storage on the server, number of accesses, etc. In addition, the system may permit multiple service providers to utilize common document management services of a server, while appearing to end-users as separate dedicated websites.

Drawing Description Text (5):
FIG. 3 depicts an illustrative hierarchy for file storage of electronic documents under the control of server computer 20;

Drawing Description Text (9):
FIG. 7 is a flowchart depicting the process of logging a storage transaction;

Drawing Description Text (11):
FIG. 9 is a flowchart depicting registration and authentication processes;

Detailed Description Text (2):
The present invention is directed to apparatus and methods for managing electronic documents over the Internet. Specifically, the present invention comprises an Internet-accessible server programmed to provide a plurality of document management services, including document storage and retrieval, collaborative file sharing and workflow services for electronic documents, an electronic document delivery service, and a document distribution service. In accordance with the principles of the present invention, these services are supported by a common database system that permits interfaces to the multiple services to be accessed using previously known web browsers, and without a specialized client application.

Detailed Description Text (4):
Referring to FIGS. 1A and 1B, illustrative architecture suitable for implementing the system and methods of the present invention is described. In FIGS. 1A and 1B, this architecture comprises personal computers 10 and 11 coupled through an open network, such as Internet 15, to document management services ("DMS") system 17. DMS system 17 comprises server computer 20, which in turn, comprises or is coupled to DMS database 25, store 30, notification server 35 and public key infrastructure server 40.

Detailed Description Text (6):
Server computer 20 is coupled to, and communicates asynchronously with, Internet 15, and includes a domain-specific digital certificate to enable secure communications. Server computer 20 preferably is programmed as a web server, e.g., to run Hyper Text Transfer Protocol ("HTTP") and with Document Management Services ("DMS") system software constructed in accordance with the present invention. In a preferred embodiment, the DMS software of the present invention runs on the web server through a Common Gateway Interface (CGI).

Detailed Description Text (7):
This enables DMS system 17 to interact with users through a web browser, rather than requiring specialized client software. In particular, a user enters information into a form displayed in a web browser. The information is transferred to server computer 20 using HTTP, and is made available to the programmed routines executing on server computer 20 through the CGI. Alternatively, the DMS software of the present invention may be implemented as "servlets," i.e., routines, typically written in the Java programming language, that run on a web server. Use of servlets also permits users to interact with DMS system 17 through a web browser.

Detailed Description Text (9):
Database 25, which may be a relational database, stores: data concerning documents controlled by server computer 20 and stored in store 30 (hereinafter, referred to

as "meta-data"), such as annotations, instructions, characteristics, etc.; user and account data; transaction data; notification data; and authorization data, all as described in greater detail hereinafter. Database 25 may be implemented on server computer 20 or on a separate computer connected to server computer 20.

Detailed Description Text (10):
Store 30 is connected to server computer 20 and stores electronic documents (or "files"). Store 30 provides a storage mechanism for storing electronic documents, and may comprise one or more hard drives, optical drives, RAIDs, etc., and further may comprise one or more stores supporting different types of storage media. Store 30 also may comprise remote storage, in which the file is stored on a remote DMS server. If multiple stores are used, DMS system 17 preferably includes a configurable algorithm to decide in which store a document will be placed, thereby evenly distributing document storage among all stores.

Detailed Description Text (11):
Store 30 preferably comprises either a relational database, where the electronic documents and information about the document is stored in the relational database, or a file system. If store 30 comprises a relational database, a unique key to the document is generated and indexed, as may be appropriate for storage of smaller files (e.g., <1 KB). If store 30 comprises a relational database, then entries in the relational database may include a storage type, a storage path (i.e., a description of location), a name, a maximum size and a state value. When store 30 comprises more than one store, the state value for each store may be set to "active" or "inactive" and documents cannot be stored in an "inactive" store. If store 30 comprises file system storage, the file system may assign a unique name to each document and the document is stored directly on the hard drive, optical drive, etc., as may be appropriate for large files.

Detailed Description Text (12):
Notification server 35, which may comprise software running on server computer 20 or on one or more separate computers connected to server computer 20, dispatches notifications, e.g., via voice message, e-mail, pager, etc., to users of DMS system 17 concerning the status of documents stored in the DMS system. Public key infrastructure server 40 ("PKI"), which also may comprise software running on server computer 20 or on one or more separate computers connected to server computer 20, provides digital certificates to users of the DMS system. The digital certificates may be used by the users to digitally sign documents for the purpose of non-repudiation.

Detailed Description Text (13):
DMS system 17 of FIG. 1A illustratively is depicted as having a single server computer 20, but also may comprise multiple server computers for use in high load scenarios. As shown in FIG. 1B, when more than one server computer is used, load balance appliance 45 may be employed to balance traffic between server computers 20A and 20B. Load balance appliance 45 may comprise software running on the server computers 20A and 20B. Alternatively, load balance appliance 45 may comprise software running on a separate computer (not shown), which is in turn connected to server computers 20A and 20B.

Detailed Description Text (14):
Referring to FIG. 2, DMS database 25 is described in greater detail. Database 25 includes a plurality of tables 61-64 and 66-68 that maintain information on documents stored in store 30. Each of tables 61-64 and 66-68 may in turn consist of multiple tables. Document information tables 61 have entries for a number of document-related parameters, including: information on a document's parent document group; information on the document instances; information on the transport method to be used for retrieval of a document instance; information on the priority of the document; expiration information: the date and time when a document instance is changed from "active" status to "archived" status; workflow information for a

document instance; security information; document rights; and document group rights.

Detailed Description Text (15):
User information tables 62 have entries for information relating to users registered to access and use the DMS system, including: the name of the user; logon information for the user, e.g., user ID and password; user notification information, e.g., notification address and transport type; billing code information; information on the user's account, where each user account is unique to a service account and user; user session information; and user group information, i.e., information on the group of users that the user is a part of, including the name of the group, the state of the group, the group's security information, and document rights for the group.

Detailed Description Text (17):
Administrative information tables 64 have entries that enable a registered user to review and track activity for a user's account, including: information on the system administrator's rights; information on logging errors; information on logging transactions; and country and language information (e.g., for a system running in the United States, the default language is English).

Detailed Description Text (18):
Notification information tables 66 maintain information necessary to generate a notification message, and include entries for: notification transport type, i.e., e-mail, facsimile, voice, or pager; information on the status of the notification, i.e. pending, sent, failed; the recipient's notification identification; priority information; and optionally, the scheduled date/time for delivery.

Detailed Description Text (19):
Transaction information tables 67 record data relating to each transaction occurring on the DMS system, and include: the identification of different transaction types; status information for each transaction; and billing information for each transaction type.

Detailed Description Text (23):
Document instances 73A, 73B and 73C correspond to specific instances of a document, and each include details about the document, a reference to the document, a document state, description, size, priority, encryption type and expiry date. The default document states are "pending," "active," "archived" and "deleted." Document states are extensible by service. A document state log is kept to track when a document instance has changed state, as described hereinbelow.

Detailed Description Text (26):
In a preferred embodiment, documents stored in the DMS system are monitored by a document state process that automatically modifies the state of a document instance based on its current state, the active date/time, and expiration date/time. States for a document instance include "pending," "active," "archived," "canceled" and "deleted." Each default state change in a document instance is logged to the DMS database, and may result, for example, in a billable transaction.

Detailed Description Text (27):
Document instances with a "pending" state have an active date/time that specifies the time at which the state of the document instance should be changed to "active." A "pending" document is not available to anyone except the Originator.

Detailed Description Text (28):
Document instances marked "active" are accessible by all Authorized Users. If a document instance has an expiration time, then the status is changed from "active" to "archived" when the expiration time is reached. At this point, document instance rights are removed for all Authorized Users except the Originator.

Detailed Description Text (29):
Document instances marked "archived" are accessible only to the Originator. The state of these documents is changed to "deleted" after a pre-determined amount of time. At this time, the physical file corresponding to the document instance is removed/deleted from storage and the corresponding document store record is deleted. Document instances marked deleted are only available for tracking and billing purposes. These document instances are removed from DMS database 25 only when the corresponding transaction log is billed and removed from database 25.

Detailed Description Text (31):
DMS system 17 also may provide a notarization feature, where each document instance is notarized by the DMS system. A digital notarization is used to authenticate an identifiable set of data at a given time. A simple notarization scheme, for example, involves creating a digital fingerprint (or digest) of a document, by using a one-way hashing algorithm, adding a timestamp, and then signing the resulting data with a private key. DMS system 17 may be configured to support multiple notarization schemes by assigning a notarization type to each digital notarization. A digital notarization object may be created, containing a reference to a document, document instance, document group, notification or transaction.

Detailed Description Text (34):
FIG. 4 is now illustratively described in overview with respect to a collaborative file sharing service of DMS system 17. In this service, an electronic document to be stored is created by an Originator using a previously known word processing, image or spreadsheet client application, and then uploaded and stored in DMS system 17. The electronic document then may be retrieved by one or more Authorized Users, as defined by the Originator during the storage process. After an Authorized User has modified the document, it is returned to store 30 of the DMS system. In accordance with the principles of the present invention, each transaction involving the document is logged in the transaction tables of DMS database 25, for example, for billing, reporting, and tracking purposes.

Detailed Description Text (35):
More particularly, at step 80, an Originator uses a previously known client application, such as a word processing, image generation application, spreadsheet, etc., to create an electronic document. Illustratively, the document may consist of a business plan and appendices for a start-up company. The Originator then connects to the Internet using his or her web browser and enters the URL for the DMS system. Once connected to the DMS system website, at step 81, the Originator initiates a user session with DMS system 17 using a logon process, described hereinbelow.

Detailed Description Text (36):
The Originator then fills out appropriate forms indicating a desire to upload the previously created electronic document to the DMS system, and at step 82 defines a list of Authorized Users who may access the document. The Originator specifies the types of access that each Authorized User is to receive, and metadata concerning the document (e.g., expiration date, etc.). Thus, for example, some Authorized Users may be granted access only to retrieve and review a document, while others are granted access to retrieve and modify the document. The specific access rights granted to each Authorized User are recorded in the document tables of DMS database 25, and the transaction is logged in the transaction tables of DMS database 25.

Detailed Description Text (37):
At step 83, the Originator requests that the document be uploaded and stored in store 30 of the DMS system. Appropriate records are generated in the document tables of DMS database 25, and the transaction is logged in the transaction tables of DMS database 25. At step 85, the document is uploaded, for example, using HTTP or FTP, and stored in store 30. During the upload process, at step 84, the document optionally may be automatically or selectively filtered in accordance with routines

appropriate for the service being performed. For example, the document may be automatically compressed or encrypted, or at the Originator's request, converted to a particular file format suitable for the Authorized Users (e.g., converted from WordPerfect.RTM. to Microsoft Word). Other forms of filtering may include formatting, translating or virus checking. Both the storage and filtering step, if performed, are logged to the appropriate tables in DMS database 25.

Detailed Description Text (38):
At step 86, notification server 35 generates notification messages to the Authorized Users informing those Users that the document is available in store 30. The notification server also may provide a notification to the Originator that the notifications to the Authorized Users have been sent or delivered, as described hereinbelow with respect to FIGS. 12A and 12B. Issuance of any notifications to the Originator and Authorized Users are logged in the Notification tables and Transaction tables of DMS database 25. At any time after the document has been stored to store 30 at step 83, the Originator may terminate his or her user session.

Detailed Description Text (39):
Once an Authorized User receives the notification that the document is available for retrieval from store 30, for example, by receipt an e-mail message or voice message, the Authorized User logs into the DMS system using a previously known web browser to create a new user session at step 87. The Authorized User may then request retrieval of the document from store 30, at step 88, and any automatic filtering, or filtering selected by the Authorized User, may be performed during the document download process at step 89. The document is then downloaded to the Authorized User at step 90. Each transaction is logged to the appropriate tables of DMS database 25.

Detailed Description Text (41):
In the context of a workflow service provided by DMS system 17, a workflow table may be associated with a document in DMS database that specifies multiple tasks to be performed in sequence by the Authorized Users. In this case, the Originator may associate or import a series of task descriptions stored in DMS database 25 with a document and a list of Authorized Users responsible for performing those tasks. After an Authorized User retrieves the document, performs the task assigned to him or her, and returns the document to store 30, notification server 35 generates and sends an appropriate notification to the Authorized User responsible for the next task in the workflow.

Detailed Description Text (42):
In the context of electronic document delivery, the Originator may specify one or multiple Authorized Users who are permitted access to the document. In this case, notification server 35 generates appropriate messages to the Authorized User(s) via the selected transport mechanism notifying those Users that the document is available in store 30. The Authorized Users may then initiate User Sessions to retrieve the document, including any specified automatic or user selected filtering requested for the document.

Detailed Description Text (45):
The Originator may "package" a document prior to uploading to the DMS system, for example, using a compression routine, encryption routine, or by adding a digital signature using applications available on the client computer, e.g., personal computer 10. Alternatively, such "packaging" may be automatically (or selectively, at the Originator's request) performed by DMS system 17 as part of a filtering process during upload and storage of the document at step 102.

Detailed Description Text (49):
At step 101, the system determines whether the Originator has specified any Authorized Users. If none are specified (or all Authorized Users have already been

confirmed), the document and metadata are stored in the DMS system at step 103, after any optional automated or requested filtering is performed at step 102. Appropriate transactions are logged to DMS database 25 at step 104 and a status message is returned to the Originator at step 105.

Detailed Description Text (50):
If the Originator specifies an Authorized User (or there are remaining Authorized Users to be confirmed), the system determines at step 106 if the specified Authorized User is registered. If so, then the DMS authorization system, described hereinafter, is updated for that Authorized User to reflect the access rights specified by the Originator or implied by that service at step 107. At step 108, the Authorized User then may be sent a notification by notification server 35 at his or her notification address. The foregoing process is repeated for each Authorized User specified by the Originator.

Detailed Description Text (56):
In case A, at step 122, the Authorized User is identified by the DMS system as a registered user. In this case, the Authorized User submits his credentials at step 123. Once the credentials are authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (57):
In case B, at step 126, the Authorized User is identified by the DMS system as a pre-registered Authorized User and the service which he or she is accessing requires an introduction number. In this case, the user is supplied with the introduction number either through a notification message (see step 112 of FIG. 5) or by the Originator using a separate channel of communication. The user then submits the introduction number at step 127. Once the introduction number is authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (59):
In case D, at step 129, the Authorized User is a pre-registered Authorized User with trusted credentials (corresponding to step 111 of FIG. 5). In this case, the pre-registered Authorized User submits the trusted credentials at 130. Once the credentials are authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (60):
In all cases, all of the Authorized User's activities are logged in the transaction log at step 125.

Detailed Description Text (61):
Transaction Logging

Detailed Description Text (62):
The DMS system of the present invention preferably supports an extensible set of transaction types. A core set of transaction types is defined by the DMS system and each service provided by the DMS system may define additional transaction types. Transaction types have the following properties: Name Billing type: "not billable"; "billable by count"; "billable by value"

Detailed Description Text (63):
Each service account may have a separate pricing plan, and each pricing plan may have an associated price per period (e.g., monthly subscription), as well as a pricing mechanism whereby each transaction type is priced for a given value of that transaction ("transaction type pricing plan"). For example, if the transaction type is document storage, then the transaction type pricing plan may include the following information: Transaction type (e.g., document storage) Pricing plan (e.g., monthly) Price (e.g., $0.50 per unit) Minimum Value (e.g., 0 KB) Maximum Value(e.g., 10 KB) Minimum chargeable price (e.g., $1) Maximum chargeable price

(e.g., $5) Visibility: Visible or Not visible, identifying whether the user can view logged information on this transaction type.

Detailed Description Text (64):
Given the foregoing information, the value of each transaction may be calculated and logged in the transaction tables of DMS database 25 with an associated price.

Detailed Description Text (65):
Each transaction may be associated to one or more of: document; document instance; document group; or a notification (i.e., a particular notification message generated by the DMS system). Each transaction also may be associated with at least one of: a user account or a service account, and preferably is timestamped with the date/time of the transaction. Additionally, each transaction may be digitally signed by the DMS system. Transactions also may be nestable, i.e., each transaction may have a parent transaction associated with it. Transactions may be used to form an audit trail for a given user, account, document, document instance, document group, or notification. Every one of these objects preferably has at least one logged transaction linked to it.

Detailed Description Text (66):
For example, for a New Document transaction in the context of a document delivery service, the following data may be stored in the transaction information tables of DMS database 25: Parent transaction=document delivery Transaction type=new document Notification ID=null Document Instance ID=9812731 Document Group ID=null Document ID=2832837 Account-ID=5632219 User ID=3878772 Amount (Value)=1 Price (Currency) =$0.50 Date/Time=12:34:43.99 EST Mar. 1, 1999 Visible=yes Status=active

Detailed Description Text (67):
The transaction links the new document to a document ID (for the document object) and a document instance ID (for the specific instance or version of this document and its related details including a pointer to its storage), the account ID, and the user ID of the user who did the transaction.

Detailed Description Text (68):
In accordance with one aspect of the present invention, the transaction log may be used to generate a billing statement for each account user. A billing statement can be generated for a particular account and particular statement period. In addition, the DMS system of the present invention also allows for user-defined identifiers (billing codes) to track and organize user activity. For example, a lawyer storing a contract on DMS system 17 may include as part of the metadata for the document an identification of the client's billing code.

Detailed Description Text (69):
During the process of generating a billing statement, the status of each of the transactions included in the billing statement are changed from "active" to "archived." Transactions marked as "archived" then may be removed from the transaction log (for improved search performance of the main transaction log) and placed into another log (e.g., an archived transactions log). Alternatively, transactions marked as archived can be automatically set to "delete" after a predetermined configurable lifetime. This status change from "archived" to "delete" may occur in both the transaction log and the archived transaction log. Transactions set to "delete" are automatically deleted after a timeout period.

Detailed Description Text (70):
Referring now to FIG. 7 the process of logging a storage transaction on the DMS system of the present invention is described. As explained above, the DMS system offers many different services, each of which may have transactions that are logged and billed. FIG. 7 is a representative example of the process of logging one such transaction.

Detailed Description Text (71):
At step 140, the logging process requires as input: the transaction value (amount), transaction type, and pricing plan. At step 141, the DMS system determines a billing type associated with the transaction type. If the transaction is "not billable," determined at step 142, the transaction price is set to zero at step 143, and transaction visibility is set according to the pricing plan at step 144. If the transaction type's billing type is "by count," determined at step 145, then the record for that range is retrieved from DMS database 25 at step 146. The transaction price is set at step 147 and visibility is set according to the pricing plan at step 148.

Detailed Description Text (72):
If the billable type is "by value", determined at step 149, then the transaction type pricing plan is retrieved from DMS database 25 at step 150. In an example in which the transaction consists of storing a 1.5 MB document to the DMS system, the transaction type is "document storage" and the value is 1.5 MB. This transaction type is billable "by value" and there are two priceable value ranges: 0-1 MB and >1 MB. The transaction type pricing plan for the first range would include the following information: Plan name (e.g. "Gold plan") Storage by size Price=$0.50 Minimum Value=0 MB Maximum Value=1 MB Minimum Chargeable Price=$0.15 Maximum Chargeable Price=Null Visibility=visible

Detailed Description Text (73):
The transaction type pricing plan for the second range would include the following information: Plan name (e.g. "Gold plan") Storage by size Price=$0.25 Minimum Value=1 MB Maximum Value=Null Minimum Chargeable Price=Null Maximum Chargeable Price=Null Visibility=visible

Detailed Description Text (74):
At step 151, the DMS system begins calculating the transaction price by setting the initial transaction price to zero. For each value range within the transaction type's value, determined at step 152, the following process is repeated: At step 153, it is determined if value >=maximum value. If so, the raw price is calculated as (maximum value--minimum value).times.price at step 154. If not, raw price is calculated as (value--minimum value).times.price at step 155. If raw price >=maximum chargeable price, determined at step 160, raw price is set to maximum chargeable price at step 161. If raw price <maximum chargeable price and if raw price <=minimum chargeable price then raw price is set to minimum chargeable price at step 163. The transaction price is set to transaction price +raw price at step 164. Therefore, continuing with the example, for a 1.5 MB file, the final transaction price would be $0.50 (1 MB.times.$0.50)+$0.125 (0.5 MB.times.$0.25) =$0.625.

Detailed Description Text (75):
After the process is repeated for each value range, the transaction price is set at step 165. Transaction visibility is set according to the pricing plan at step 166, and all of the information is logged into the transaction log, completing the logging process.

Detailed Description Text (79):
The services interfaces also permit users to interact with DMS system 17 using client applications specific to the service to be performed. For example, a web browser may be used to make requests to DMS system 17 using HTTP over Secure Sockets Layers (SSL) protocol, and a response may be returned in Hyper Text Markup Language ("HTML"). A word processor application may make a request to DMS system 17 using HTTP over SSL and a response may be returned in Extensible Markup Language ("XML"). Each DMS service may respond to requests for data using different formats, e.g., HTML, XML, etc. A DMS service also may respond to requests by structuring the data differently according to a service provider's preferences.

Detailed Description Text (80):
Service providers 167a-167c in FIG. 8 each provide a DMS service using DMS system 17. In accordance with one aspect of the present invention, DMS system 17 may include customization functions, that permit different service providers to access a single DMS system, but create the appearance of separate dedicated server computers. For example, by accessing a document delivery service with a service account provided by ACME Document Delivery, the user will view ACME's corporate logo in the data returned. This may be accomplished, for example, using a "logo" parameter, stored in account information tables 64 (see FIG. 2), which identifies a particular service provider's corporate information to be displayed to a user account administered by that service provider. There may be one or many service providers 167a-c for each of one or more services on a single DMS system.

Detailed Description Text (83):
The user registration and authentication processes for registering as a user of DMS system 17 are now described. As described hereinabove, many of the services offered by the DMS system of the present invention require a user to have a user account, and information on each user account is stored in the account information tables of DMS database 25. In a referred embodiment, a user may obtain a user account either by: 1) registration and authentication or 2) through introduction by another registered user.

Detailed Description Text (84):
Each user account is unique to a service account and user; information on each service account also is stored in DMS database 25. A service account comprises a service, a service provider, a pricing plan for every transaction the user does with an account, a limit plan that limits the use of an account (e.g., a limit on the maximum file size that can be uploaded into the DMS system), a feature plan for customizing the features available for each service (e.g., disabling the scheduled delivery feature of the document delivery service), and billing information (billing address and payment information).

Detailed Description Text (85):
Referring to FIG. 9 the registration and authentication processes used by a user to gain access to DMS system 17 are described. At step 170, the registrant accesses the DMS system registration interface, for example, using a web browser to access the DMS system's registration interface URL. Next, at step 171, the registrant selects a DMS service for which he or she wishes to be registered. At step 172, the DMS determines whether the registrant already has an existing DMS service account.

Detailed Description Text (86):
If the registrant already has a DMS account, registration for a new service requires that the registrant provide his or her user credentials at step 173 and then authenticate those credentials at step 180. If the registrant has no pre-existing account, determined at step 172, the registrant is requested to provide personal information, such as name, address, notification address (e.g., e-mail address, telephone number, IP address), payment information, etc. at step 174. At step 175, the DMS server computer processes and verifies the registration information. If the information is not successfully verified at step 176, the registrant is informed that insufficient information has been provided, at step 186, and the registrant is requested to resubmit the information.

Detailed Description Text (87):
If the information is successfully verified, the registrant is provided with user credentials over a secure link at step 177. User credentials, which may consist, for example, of alphanumeric user IDs, alphanumeric passwords, digital certificates, and/or notification addresses, permit the user to securely access documents, upload documents, view authorized information on documents, digitally sign documents, etc. A user's credentials uniquely identify the user to the DMS system. At step 178, the registrant is given instructions to authenticate his or

her credentials.

Detailed Description Text (88):
Once the registrant is issued credentials, or is determined to already have credentials, the authentication process begins, at step 179. This may be accomplished by the registrant accessing the DMS authentication interface by inputting the URL associated with the DMS authentication interface into his or her web browser. Once the registrant is successfully authenticated, at step 180, the registrant's new service account is ready for use at step 181. If the registrant is not successfully authenticated at step 180, an authentication failure is logged at step 182. If the number of authentication failures exceeds a predetermined number, at step 183, the registrant's ability to authenticate is locked for a predetermined period of time at step 184. If the number of authentication failures does not exceed the predetermined safety limit, the registrant is prompted to authenticate again at step 185.

Detailed Description Text (89):
Referring now to FIG. 10, after a user has become registered and has authenticated his or her credentials with the DMS system, the user then may access the services provided by the DMS service by logging on to the DMS system. A user first accesses the DMS logon service at step 190, for example, using a web browser to access the URL associated with the DMS logon service. The user then supplies his or her credentials at step 191, and the DMS checks to see if the credentials are valid at step 192. If the credentials are valid, a user session is created at step 193 and the user is given access to the DMS system at step 194.

Detailed Description Text (92):
HTTP sessions are stateless, so information on these sessions must be maintained in database 25. Communications to server 20 contains a session identifier number that references session information in database 25. Sessions are managed by an automatic process, illustrated in FIG. 11, that continually monitors the length of a session to determine if a current session is longer than a specific, predetermined interval. If there is an active session, determined at step 200, the DMS system determines if the session length is greater than the predetermined interval, at step 201. If the interval has been exceeded, the user session is rendered inactive at step 202 and a flag to that effect is entered in the corresponding database entry. The process is repeated at step 203 for each active session. Alternatively, a user forced logout/exit also may render a user session inactive and the corresponding database entry is flagged accordingly.

Detailed Description Text (94):
Referring now to FIGS. 12A and 12B, the notification request and confirmation services available on a preferred embodiment of DMS system 17 are described. Notification messages are generated by notification server 35 in response to various user events. For example, when a registrant registers for a DMS service, the registrant receives a notification with instructions on authorization, as discussed hereinabove with respect to step 178 of FIG. 9.

Detailed Description Text (96):
With respect to FIG. 12A, the notification request process performed by DMS system 17 is described. At step 210, a notification message is created by notification server 35 responsive to some user-initiated event. At step 211, a notification request is created that contains some or all of the following information: (1) the subject of the message; (2) the Originator's notification address (e.g., an e-mail address); (3) the notification address of the Authorized User(s); (4) the priority of the notification (e.g., high, medium, or low); (5) the body of the message, including a unique notification identifier created by the DMS system; (6) optionally, an indication of the date and time that the message should be delivered; (7) a status flag (e.g., "pending", "sent", or "failed") indicating the status of the notification delivery, initially set to "pending"; (8) the transport

type for the notification (e.g., e-mail, voice message, etc.); and (9) a retry counter that tracks the number of times that a notification request has been processed (initially set to zero, and incremented upon each unsuccessful delivery attempt until the notification request status is marked "failed.") The notification request is queued, at step 212, with a status of "pending," in notification information tables 66 of DMS database 25.

Detailed Description Text (97):
The notification delivery process is described with respect to FIG. 12B. At step 220, the system iterates through the records in the tables with a "pending" flag. At step 221, notification server 35 attempts to deliver the notification using the specified transport system for that Authorized User. DMS system 17 then checks to see if there is a transport rejection, at step 222, for example, if notification server 35 is not working. If no transport rejection is detected, the notification request flag is set to "sent" at step 223, and the notification transaction is logged as sent at step 224.

Detailed Description Text (98):
If a transport rejection is detected, at step 222, a retry counter is checked at step 225. If the number of retries does not exceed a predetermined limit, the retry counter is incremented at step 226 and the notification process begins again. If the number of retries exceeds the predetermined limit, the notification request flag is set to "failed" at step 227, and the notification transaction is logged as "failed," at step 228. At step 229, the DMS system checks information on the origin of the notification request; where the origin of a notification request may be either the DMS system or a system user.

Detailed Description Text (100):
It is possible for a notification to be sent, but for the send to be unsuccessful, for example, if the notification recipient's e-mail address is incorrect. For this reason, each notification transport that delivers notification messages also preferably receives messages that notifications have been sent successfully but have failed during transport. Each notification transport is polled by an automated process for any new messages. Upon receiving a failed notification, this process determines (if possible) the notification identifier, marks the original notification request as "failed" and logs a failed notification transaction linked to the original notification request. In addition, if the origin is not the DMS system, a notification is generated and sent to the sender indicating a failed delivery.

Other Reference Publication (4):
Borenstein, Nathaniel et al., A Multi-media Message System for Andrew, USENIX Winter Conference, Dallas, TX, pp. 37-42 (Feb. 9-12, 1988).

Other Reference Publication (7):
Kent, Jack et al., Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS, Proceedings of the Fourteenth International Conference on Very Large Data Bases, Los Angeles, CA, pp 112-123 (1988).

Other Reference Publication (18):
Overview of the Trans-Virtual Enterpriser Server, Product Overview.

CLAIMS:

1. An Internet-based document management system comprising: an Internet-based store for storing an electronic document; a database programmed to include and access a document table for storing information about the electronic document and a transaction table that stores information about transactions performed on the electronic document by different users of the Internet-based document management system; and a server connected to the Internet-based store and the database, the

server programmed to receive the document from a remote computer using an Internet protocol and store the document in the Internet-based store, wherein the server is programmed to provide a plurality of services supported by the database, including filtering the electronic document before storing the document in the Internet-based store using one or more of: compression, decompression, encryption, decryption, translation, and formatting.

7. The Internet-based document management system of claim 1 wherein the database further comprises an account information table including accounting data, and the server is programmed to apply the accounting data to the information stored in the transaction table to determine a price reflecting usage of the document management system.

8. The Internet-based document management system of claim 1 further comprising a notification server that generates and dispatches notifications containing information about the electronic document using information stored in the database.

9. A method of providing Internet-based document management comprising: providing an Internet-based store, a database and a server connected to the Internet-based store and the database; accepting a connection from a first remote computer to the server using an Internet protocol; receiving an uploaded electronic document from the first remote computer to the server using an Internet protocol; generating a record in a document table of the database to store information about the electronic document; generating a record in a transaction table of the database to store information about transactions performed on the electronic document; filtering the electronic document by applying one or more of: compression, decompression, encryption, decryption, translation and formatting to the document; storing the electronic document in the Internet-based store; accepting a connection from a second remote computer to the server using an Internet protocol; and providing to the second remote computer a plurality of document management services supported by the database; wherein the transaction table stores information about transactions performed on the electronic document by multiple users.

13. The method of claim 9 wherein the database further comprises an account information table including accounting data, the method further comprising applying the accounting data to the record in the transaction table to determine a price.

16. The method of claim 9 further comprising generating and dispatching a notification from the server to the second remote computer containing information about the electronic document using information stored in the database.

L53: Entry 8 of 12 File: USPT Nov 6, 2001

DOCUMENT-IDENTIFIER: US 6314425 B1
TITLE: Apparatus and methods for use of access tokens in an internet document management system

Abstract Text (1):
An Internet-based document management system and methods are provided wherein access to the system and its services may be controlled through use of access tokens. The Internet-based document management system allows an electronic document to be stored on an Internet-accessible server and accessed using a previously known web browser, downloaded for review or manipulation, and then returned to the server for access by further users. The server is programmed to generate and validate access tokens and provide a plurality of services supported by a common database and document store, including storage and retrieval services, an electronic document delivery service, a document distribution service, a collaborative file sharing service and a workflow service. The system preferably also is programmed with a security function, a filtering function, accounting functions that enable detailed accounting of transactions occurring on the system, and a customization function that permits multiple service providers to utilize the common document management services of a server, while presenting end-users with distinct dedicated websites.

Brief Summary Text (4):
Document management systems are known that permit multiple users to store and retrieve electronic documents on a closed client/server architecture network, such as a local area network or wide area network. These previously known document management systems, such as DOCSFusion, available from PCDOCS, Inc., Toronto, Ontario, Canada and EDMS 98, available from Documentum, Inc., Pleasanton, Calif., require the presence of a client application on each node of the network that is to access and manipulate files.

Brief Summary Text (6):
Smith U.S. Pat. No. 5,790,790 describes an Internet electronic document delivery system, wherein an e-mail message contains a direct reference (i.e., a Uniform Resource Locator or "URL") to an electronic document stored on a server. When a recipient receives the e-mail message, the direct reference is used to access the document. A drawback of the system described in that patent, is that the sending computer must include a specialized client application for interacting with the server. The system described in that patent also lacks the kinds of transaction logging and accounting functions needed to provide a useful document management system.

Brief Summary Text (7):
The POSTA.RTM. system, offered by Tumbleweed Software Corporation, Redwood City, Calif., overcomes some of the drawbacks of the system described in the foregoing patent. For example, the POSTA.RTM. system eliminates the need for specialized client software for basic document delivery operations, and permits the use of a previously known web browser, such as Internet Explorer 4.0.RTM., available from Microsoft Corp., Redmond, Wash., or Netscape Navigator.RTM., Netscape Corporation, Mountain View, Calif. That commercial system also eliminates use of the direct reference in the e-mail message, instead providing a URL for a webpage that

provides the user with several options for document delivery. The system provides none of the capabilities normally associated with a document management system.

Brief Summary Text (8):
Higley U.S. Pat. No. 5,790,793, like the foregoing Smith patent, also describes an Internet electronic document delivery system wherein an e-mail message includes a URL reference to a document stored in a server. This system described in this patent also requires the use of a specialized client application, and is limited to an electronic document delivery service.

Brief Summary Text (11):
In view of the foregoing, it would be desirable to provide a document management system and methods that permit electronic documents to be made available for use on open systems, such as the Internet, and to be accessed using a previously known web browser--without the need for a specialized client application.

Brief Summary Text (15):
It yet further would be desirable to provide an Internet-based document management system and methods that enable the transaction logging and accounting functions needed for multi-user collaborative electronic document manipulation, for example, so that revisions to a document may be tracked.

Brief Summary Text (16):
It also would be desirable to provide an Internet-based document management system and methods that enable tracking of transactions performed on a document for billing purposes, and which provide needed access-control protocols, for example, so that specific users' privileges with respect to a document may be defined.

Brief Summary Text (19):
In view of the foregoing, it is an object of this invention to provide a document management system and methods that permit electronic documents to be made available for use on open systems, such as the Internet, and to be accessed using a previously known web browser--without the need for a specialized client application.

Brief Summary Text (23):
It is a further object of this invention to provide an Internet-based document management system and methods that enable the transaction logging and accounting capabilities needed for multi-user collaborative electronic document manipulation, for example, so that revisions to a document may be tracked.

Brief Summary Text (24):
It is a still further object of the present invention to provide an Internet-based document management system and methods that enable tracking of transactions performed on a document for billing purposes, and which provide needed access-control protocols, for example, so that specific users' privileges with respect to a document may be defined.

Brief Summary Text (26):
These and other objects of the present invention are accomplished by providing an Internet-based document management system and methods wherein an electronic document may be stored on an Internet-accessible server and accessed using a previously known web browser, downloaded for review or manipulation, and then returned to the server for access by further users. The server is programmed with several routines that perform numerous functions, referred to hereinafter as "services," that provide a full-featured document management system. In accordance with the principles of the present invention, the server is programmed to generate access tokens, derived from random numbers independent of user or resource information, and these tokens are used to limit a user's access to the services provided by the document management system.

Brief Summary Text (27):
In a preferred embodiment, the document management system is programmed to provide a plurality of services supported by a common database and document store. These services preferably include storage and retrieval services to and from an Internet-based storage site, an electronic document delivery service, a collaborative file sharing service and a workflow service, and a document distribution service. The server also preferably is programmed to generate access tokens from random numbers, and verify access tokens as a security function, to verify or define a requestor's ability to access an electronic document, a filtering function that performs selective or automatic filtering of documents during storage to and/or retrieval from the storage site, and accounting functions that enable detailed accounting of, for example, usage of storage on the server, number of accesses, etc. In addition, the system may permit multiple service providers to utilize common document management services of a server, while appearing to end-users as separate dedicated websites.

Drawing Description Text (9):
FIG. 7 is a flowchart depicting the process of logging a storage transaction;

Drawing Description Text (11):
FIG. 9 is a flowchart depicting registration and authentication processes;

Detailed Description Text (2):
The present invention is directed to apparatus and methods for the use of access tokens in a system for managing electronic documents over the Internet. Specifically, the present invention comprises an Internet-accessible server programmed to generate access tokens and provide a plurality of document management services, including document storage and retrieval, collaborative file sharing and workflow services for electronic documents, an electronic document delivery service, and a document distribution service. As used herein, the term "access token" refers to a security code used to restrict a user's access to the DMS system of the present invention, which access token is comprised of a signed string, unique to a transaction and generated from one or more random numbers independent of any user information or resource information or any other identifiable information. Further, in accordance with the principles of the present invention, these services are supported by a common database system that is used to validate access tokens and permits interfaces to the multiple services to be accessed using previously known web browsers, and without a specialized client application.

Detailed Description Text (4):
Referring to FIGS. 1A and 1B, illustrative architecture suitable for implementing the system and methods of the present invention is described. In FIGS. 1A and 1B, this architecture comprises personal computers 10 and 11 coupled through an open network, such as Internet 15, to document management services ("DMS") system 17. DMS system 17 comprises server computer 20, which in turn, comprises or is coupled to DMS database 25, store 30, notification server 35 and public key infrastructure server 40.

Detailed Description Text (6):
Server computer 20 is coupled to, and communicates asynchronously with, Internet 15, and includes a domain-specific digital certificate to enable secure communications. Server computer 20 preferably is programmed as a web server, e.g., to run Hyper Text Transfer Protocol ("HTTP") and with Document Management Services ("DMS") system software constructed in accordance with the present invention. In a preferred embodiment, the DMS software of the present invention runs on the web server through a Common Gateway Interface (CGI).

Detailed Description Text (7):
This enables DMS system 17 to interact with users through a web browser, rather

than requiring specialized client software. In particular, a user enters information into a form displayed in a web browser. The information is transferred to server computer 20 using HTTP, and is made available to the programmed routines executing on server computer 20 through the CGI. Alternatively, the DMS software of the present invention may be implemented as "servlets," i.e., routines, typically written in the Java programming language, that run on a web server. Use of servlets also permits users to interact with DMS system 17 through a web browser.

Detailed Description Text (9):
Database 25, which may be a relational database, stores: data concerning documents controlled by server computer 20 and stored in store 30 (hereinafter, referred to as "meta-data"), such as annotations, instructions, characteristics, etc.; user and account data; transaction data; notification data; and authorization data, all as described in greater detail hereinafter. Database 25 may be implemented on server computer 20 or on a separate computer connected to server computer 20.

Detailed Description Text (10):
Store 30 is connected to server computer 20 and stores electronic documents (or "files"). Store 30 provides a storage mechanism for storing electronic documents, and may comprise one or more hard drives, 30 optical drives, RAIDs, etc., and further may comprise one or more stores supporting different types of storage media. Store 30 also may comprise remote storage, in which the file is stored on a remote DMS server. If multiple stores are used, DMS system 17 preferably includes a configurable algorithm to decide in which store a document will be placed, thereby evenly distributing document storage among all stores.

Detailed Description Text (11):
Store 30 preferably comprises either a relational database, where the electronic documents and information about the document is stored in the relational database, or a file system. If store 30 comprises a relational database, a unique key to the document is generated and indexed, as may be appropriate for storage of smaller files (e.g., <1 KB). If store 30 comprises a relational database, then entries in the relational database may include a storage type, a storage path (i.e., a description of location), a name, a maximum size and a state value. When store 30 comprises more than one store, the state value for each store may be set to "active" or "inactive" and documents cannot be stored in an "inactive" store. If store 30 comprises file system storage, the file system may assign a unique name to each document and the document is stored directly on the hard drive, optical drive, etc., as may be appropriate for large files.

Detailed Description Text (12):
Notification server 35, which may comprise software running on server computer 20 or on one or more separate computers connected to server computer 20, dispatches notifications, e.g., via voice message, e-mail, pager, etc., to users of DMS system 17 concerning the status of documents stored in the DMS system. Public key infrastructure server 40 ("PKI"), which also may comprise software running on server computer 20 or on one or more separate computers connected to server computer 20, provides digital certificates to users of the DMS system. The digital certificates may be used by the users to digitally sign documents for the purpose of non-repudiation.

Detailed Description Text (13):
DMS system 17 of FIG. 1A illustratively is depicted as having a single server computer 20, but also may comprise multiple server computers for use in high load scenarios. As shown in FIG. 1B, when more than one server computer is used, load balance appliance 45 may be employed to balance traffic between server computers 20A and 20B. Load balance appliance 45 may comprise software running on the server computers 20A and 20B. Alternatively, load balance appliance 45 may comprise software running on a separate computer (not shown), which is in turn connected to server computers 20A and 20B.

Detailed Description Text (15):
Document information tables 61 have entries for a number of document-related parameters, including: information on a document's parent document group; information on the document instances; information on the transport method to be used for retrieval of a document instance; information on the priority of the document; expiration information: the date and time when a document instance is changed from "active" status to "archived" status; workflow information for a document instance; security information; document rights; and document group rights.

Detailed Description Text (16):
User information tables 62 have entries for information relating to users registered to access and use the DMS system, including: the name of the user; logon information for the user, e.g., user ID and password; user notification information, e.g., notification address and transport type; billing code information; information on the user's account, where each user account is unique to a service account and user; user session information; and user group information, i.e., information on the group of users that the user is a part of, including the name of the group, the state of the group, the group's security information, and document rights for the group.

Detailed Description Text (19):
Administrative information tables 64 have entries that enable a registered user to review and track activity for a user's account, including: information on the system administrator's rights; information on logging errors; information on logging transactions; and country and language information (e.g., for a system running in the United States, the default language is English).

Detailed Description Text (20):
Notification information tables 66 maintain information necessary to generate a notification message, and include entries for: notification transport type, i.e., e-mail, facsimile, voice, or pager; information on the status of the notification, i.e. pending, sent, failed; the recipient's notification identification; priority information; and optionally, the scheduled date/time for delivery.

Detailed Description Text (21):
Transaction information tables 67 record data relating to each transaction occurring on the DMS system, and include: the identification of different transaction types; status information for each transaction; and billing information for each transaction type.

Detailed Description Text (24):
Document objects 72A and 72B represent a generalized high level description of a document, and include a document name, document type, description, expiry date, document state, account ID, billing code ID and document ID. Document objects also may have extensible property types.

Detailed Description Text (25):
Document instances 73A, 73B and 73C correspond to specific instances of a document, and each include details about the document, a reference to the document stored in Store 30, a parent document object ID, a type, account ID, billing code ID, a document state, document instance ID, description, size, priority, and encryption type. The default document states are "pending," "active," "archived" and "deleted." Document states are extensible by service. A document state log is kept to track when a document instance has changed state, as described hereinbelow.

Detailed Description Text (28):
In a preferred embodiment, documents stored in the DMS system are monitored by a document state process that automatically modifies the state of a document instance

based on its current state, the active date/time, and expiration date/time. States for a document instance include "pending," "active," "archived," "canceled" and "deleted." Each default state change in a document instance is logged to the DMS database, and may result, for example, in a billable transaction.

Detailed Description Text (29):
Document instances with a "pending" state have an active date/time that specifies the time at which the state of the document instance should be changed to "active." A "pending" document is not available to anyone except the Originator.

Detailed Description Text (30):
Document instances marked "active" are accessible by all Authorized Users. If a document instance has an expiration time, then the status is changed from "active" to "archived" when the expiration time is reached. At this point, document instance rights are removed for all Authorized Users except the Originator.

Detailed Description Text (31):
Document instances marked "archived" are accessible only to the Originator. The state of these documents is changed to "deleted" after a predetermined amount of time. At this time, the electronic document (physical file) corresponding to the document instance is removed/deleted from storage and the corresponding document store record is deleted. Document instances marked deleted are only available for tracking and billing purposes. These document instances are removed from DMS database 25 only when the corresponding transaction log is billed and removed from database 25.

Detailed Description Text (33):
DMS system 17 also may provide a notarization feature, where each document instance is notarized by the DMS system. A digital notarization is used to authenticate an identifiable set of data at a given time. A simple notarization scheme, for example, involves creating a digital fingerprint (or digest) of a document, by using a one-way hashing algorithm, adding a timestamp, and then signing the resulting data with a private key. DMS system 17 may be configured to support multiple notarization schemes by assigning a notarization type to each digital notarization. A digital notarization object may be created, containing a reference to a document, document instance, document group, notification or transaction.

Detailed Description Text (36):
FIG. 4 is now illustratively described in overview with respect to a collaborative file sharing service of DMS system 17. In this service, an electronic document to be stored is created by an Originator using a previously known word processing, image or spreadsheet client application, and then uploaded and stored in DMS system 17. The electronic document then may be retrieved by one or more Authorized Users, as defined by the Originator during the storage process. After an Authorized User has modified the document, it is returned to store 30 of the DMS system. In accordance with the principles of the present invention, each transaction involving the document is logged in the transaction tables of DMS database 25, for example, for billing, reporting, and tracking purposes.

Detailed Description Text (37):
More particularly, at step 80, an Originator uses a previously known client application, such as a word processing, image generation application, spreadsheet, etc., to create an electronic document. Illustratively, the document may consist of a business plan and appendices for a start-up company. The Originator then connects to the Internet using his or her web browser and enters the URL for the DMS system. Once connected to the DMS system website, at step 81, the Originator initiates a user session with DMS system 17 using a logon process, described hereinbelow.

Detailed Description Text (38):
The Originator then fills out appropriate forms indicating a desire to upload the

previously created electronic document to the DMS system, and at step 82 defines a list of Authorized Users who may access the document. The Originator specifies the types of access that each Authorized User is to receive, and metadata concerning the document (e.g., expiration date, etc.). Thus, for example, some Authorized Users may be granted access only to retrieve and review a document, while others are granted access to retrieve and modify the document. The specific access rights granted to each Authorized User are recorded in the document tables of DMS database 25, and the transaction is logged in the transaction tables of DMS database 25.

Detailed Description Text (39):
At step 83, the Originator requests that the document be uploaded and stored in store 30 of the DMS system. Appropriate records are generated in the document tables of DMS database 25, and the transaction is logged in the transaction tables of DMS database 25. At step 85, the document is uploaded, for example, using HTTP or FTP, and stored in store 30. During the upload process, at step 84, the document optionally may be automatically or selectively filtered in accordance with routines appropriate for the service being performed. For example, the document may be automatically compressed or encrypted, or at the Originator's request, converted to a particular file format suitable for the Authorized Users (e.g., converted from WordPerfect.RTM. to Microsoft Word). Other forms of filtering may include formatting, translating or virus checking. Both the storage and filtering step, if performed, are logged to the appropriate tables in DMS database 25.

Detailed Description Text (40):
At step 86, notification server 35 generates notification messages to the Authorized Users informing those Users that the document is available in store 30. The notification server also may provide a notification to the Originator that the notifications to the Authorized Users have been sent or delivered, as described hereinbelow with respect to FIGS. 12A and 12B. Issuance of any notifications to the Originator and Authorized Users are logged in the Notification tables and Transaction tables of DMS database 25. At any time after the document has been stored to store 30 at step 83, the Originator may terminate his or her user session.

Detailed Description Text (41):
Once an Authorized User receives the notification that the document is available for retrieval from store 30, for example, by receipt of an e-mail message or voice message, the Authorized User logs into the DMS system using a previously known web browser to create a new user session at step 87. The Authorized User may then request retrieval of the document from store 30, at step 88, and any automatic filtering, or filtering selected by the Authorized User, may be performed during the document download process at step 89. The document is then downloaded to the Authorized User at step 90. Each transaction is logged to the appropriate tables of DMS database 25.

Detailed Description Text (43):
In the context of a workflow service provided by DMS system 17, a workflow table may be associated with a document in DMS database that specifies multiple tasks to be performed in sequence by the Authorized Users. In this case, the Originator may associate or import a series of task descriptions stored in DMS database 25 with a document and a list of Authorized Users responsible for performing those tasks. After an Authorized User retrieves the document, performs the task assigned to him or her, and returns the document to store 30, notification server 35 generates and sends an appropriate notification to the Authorized User responsible for the next task in the workflow.

Detailed Description Text (44):
In the context of electronic document delivery, the Originator may specify one or multiple Authorized Users who are permitted access to the document. In this case, notification server 35 generates appropriate messages to the Authorized User(s) via

the selected transport mechanism notifying those Users that the document is available in store 30. The Authorized Users may then initiate User Sessions to retrieve the document, including any specified automatic or user selected filtering requested for the document.

Detailed Description Text (47):
The Originator may "package" a document prior to uploading to the DMS system, for example, using a compression routine, encryption routine, or by adding a digital signature using applications available on the client computer, e.g., personal computer 10. Alternatively, such "packaging" may be automatically (or selectively, at the Originator's request) performed by DMS system 17 as part of a filtering process during upload and storage of the document at step 102.

Detailed Description Text (51):
At step 101, the system determines whether the Originator has specified any Authorized Users. If none are specified (or all Authorized Users have already been confirmed), the document and metadata are stored in the DMS system at step 103, after any optional automated or requested filtering is performed at step 102. Appropriate transactions are logged to DMS database 25 at step 104 and a status message is returned to the Originator at step 105.

Detailed Description Text (52):
If the Originator specifies an Authorized User (or there are remaining Authorized Users to be confirmed), the system determines at step 106 if the specified Authorized User is registered. If so, then the DMS authorization system, described hereinafter, is updated for that Authorized User to reflect the access rights specified by the Originator or implied by that service at step 107. At step 108, the Authorized User then may be sent a notification by notification server 35 at his or her notification address. The foregoing process is repeated for each Authorized User specified by the originator.

Detailed Description Text (61):
In case A, at step 122, the Authorized User is identified by the DMS system as a registered user. In this case, the Authorized User submits his credentials at step 123. Once the credentials are authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (62):
In case B, at step 126, the Authorized User is identified by the DMS system as a pre-registered Authorized User and the service which he or she is accessing requires an introduction number. In this case, the user is supplied with the introduction number either through a notification message (see step 112 of FIG. 5) or by the Originator using a separate channel of communication. The user then submits the introduction number at step 127. Once the introduction number is authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (64):
In case D, at step 129, the Authorized User is a pre-registered Authorized User with trusted credentials (corresponding to step 111 of FIG. 5). In this case, the pre-registered Authorized User submits the trusted credentials at 130. Once the credentials are authenticated, the user is provided access to the documents and data at step 124.

Detailed Description Text (65):
In all cases, all of the Authorized User's activities are logged in the transaction log at step 125.

Detailed Description Text (66):
Transaction Logging

Detailed Description Text (67):
The DMS system of the present invention preferably supports an extensible set of transaction types. A core set of transaction types is defined by the DMS system and each service provided by the DMS system may define additional transaction types. Transaction types have the following properties:

Detailed Description Text (70):
Each service account may have a separate pricing plan, and each pricing plan may have an associated price per period (e.g., monthly subscription), as well as a pricing mechanism whereby each transaction type is priced for a given value of that transaction ("transaction type pricing plan"). For example, if the transaction type is document storage, then the transaction type pricing plan may include the following information:

Detailed Description Text (71):
Transaction type (e.g., document storage)

Detailed Description Text (78):
Visibility: Visible or Not visible, identifying whether the user can view logged information on this transaction type.

Detailed Description Text (79):
Given the foregoing information, the value of each transaction may be calculated and logged in the transaction tables of DMS database 25 with an associated price.

Detailed Description Text (80):
Each transaction may be associated to one or more of: document; document instance; document group; or a notification (i.e., a particular notification message generated by the DMS system). Each transaction also may be associated with at least one of: a user account or a service account, and preferably is timestamped with the date/time of the transaction. Additionally, each transaction may be digitally signed by the DMS system. Transactions also may be nestable, i.e., each transaction may have a parent transaction associated with it. Transactions may be used to form an audit trail for a given user, account, document, document instance, document group, or notification. Every one of these objects preferably has at least one logged transaction linked to it.

Detailed Description Text (81):
For example, for a New Document transaction in the context of a document delivery service, the following data may be stored in the transaction information tables of DMS database 25:

Detailed Description Text (82):
Parent transaction=document delivery

Detailed Description Text (83):
Transaction type=new document

Detailed Description Text (94):
Status=active

Detailed Description Text (95):
The transaction links the new document to a document ID (for the document object) and a document instance ID (for the specific instance or version of this document and its related details including a pointer to its storage), the account ID, and the user ID of the user who did the transaction.

Detailed Description Text (96):
The transaction log may be used to generate a billing statement for each account user. A billing statement can be generated for a particular account and particular

statement period. In addition, the DMS system of the present invention also allows for user-defined identifiers (billing codes) to track and organize user activity. For example, a lawyer storing a contract on DMS system 17 may include as part of the metadata for the document an identification of the client's billing code.

Detailed Description Text (97):
During the process of generating a billing statement, the status of each of the transactions included in the billing statement are changed from "active" to "archived." Transactions marked as "archived" then may be removed from the transaction log (for improved search performance of the main transaction log) and placed into another log (e.g., an archived transactions log). Alternatively, transactions marked as archived can be automatically set to "delete" after a predetermined configurable lifetime. This status change from "archived" to "delete" may occur in both the transaction log and the archived transaction log. Transactions set to "delete" are automatically deleted after a timeout period.

Detailed Description Text (98):
Referring now to FIG. 7, the process of logging a storage transaction on the DMS system of the present invention is described. As explained above, the DMS system offers many different services, each of which may have transactions that are logged and billed. FIG. 7 is a representative example of the process of logging one such transaction.

Detailed Description Text (99):
At step 140, the logging process requires as input: the transaction value (amount), transaction type, and pricing plan. At step 141, the DMS system determines a billing type associated with the transaction type. If the transaction is "not billable," determined at step 142, the transaction price is set to zero at step 143, and transaction visibility is set according to the pricing plan at step 144. If the transaction type's billing type is "by count," determined at step 145, then the record for that range is retrieved from DMS database 25 at step 146. The transaction price is set at step 147 and visibility is set according to the pricing plan at step 148.

Detailed Description Text (100):
If the billable type is "by value", determined at step 149, then the transaction type pricing plan is retrieved from DMS database 25 at step 150. In an example in which the transaction consists of storing a 1.5 MB document to the DMS system, the transaction type is "document storage" and the value is 1.5 MB. This transaction type is billable "by value" and there are two priceable value ranges: 0-1 MB and >1 MB. The transaction type pricing plan for the first range would include the following information:

Detailed Description Text (109):
The transaction type pricing plan for the second range would include the following information:

Detailed Description Text (118):
At step 151, the DMS system begins calculating the transaction price by setting the initial transaction price to zero. For each value range within the transaction type's value, determined at step 152, the following process is repeated: At step 153, it is determined if value>=maximum value. If so, the raw price is calculated as (maximum value-minimum value).times.price at step 154. If not, raw price is calculated as (value-minimum value).times.price at step 155. If raw price>=maximum chargeable price, determined at step 160, raw price is set to maximum chargeable price at step 161. If raw price<maximum chargeable price and if raw price<=minimum chargeable price then raw price is set to minimum chargeable price at step 163. The transaction price is set to transaction price+raw price at step 164. Therefore, continuing with the example, for a 1.5 MB file, the final transaction price would be $0.50 (1 MB.times.$0.50)+$0.125 (0.5 MB.times.$0.25)=$0.625.

Detailed Description Text (119):
After the process is repeated for each value range, the transaction price is set at step 165. Transaction visibility is set according to the pricing plan at step 166, and all of the information is logged into. the transaction log, completing the logging process.

Detailed Description Text (123):
The services interfaces also permit users to interact with DMS system 17 using client applications specific to the service to be performed. For example, a web browser may be used to make requests to DMS system 17 using HTTP over Secure Sockets Layers (SSL) protocol, and a response may be returned in Hyper Text Markup Language ("HTML"). A word processor application may make a request to DMS system 17 using HTTP over SSL and a response may be returned in Extensible Markup Language ("XML"). Each DMS service may respond to requests for data using different formats, e.g., HTML, XML, etc. A DMS service also may respond to requests by structuring the data differently according to a service provider's preferences.

Detailed Description Text (124):
Service providers 167a-167c in FIG. 8 each provide a DMS service using DMS system 17. In accordance with one aspect of the present invention, DMS system 17 may include customization functions, that permit different service providers to access a single DMS system, but create the appearance of separate dedicated server computers. For example, by accessing a document delivery service with a service account provided by ACME Document Delivery, the user will view ACME's corporate logo in the data returned. This may be accomplished, for example, using a "logo" parameter, stored in account information tables 64 (see FIG. 2), which identifies a particular service provider's corporate information to be displayed to a user account administered by that service provider. There may be one or many service providers 167a-c for each of one or more services on a single DMS system.

Detailed Description Text (127):
The user registration and authentication processes for registering as a user of DMS system 17 are now described. As described hereinabove, many of the services offered by the DMS system of the present invention require a user to have a user account, and information on each user account is stored in the account information tables of DMS database 25. In a preferred embodiment, a user may obtain a user account either by: 1) registration and authentication or 2) through introduction by another registered user.

Detailed Description Text (128):
Each user account is unique to a service account and user; information on each service account also is stored in DMS database 25. A service account comprises a service, a service provider, a pricing plan for every transaction the user does with an account, a limit plan that limits the use of an account (e.g., a limit on the maximum file size that can be uploaded into the DMS system), a feature plan for customizing the features available for each service (e.g., disabling the scheduled delivery feature of the document delivery service), and billing information (billing address and payment information).

Detailed Description Text (129):
Referring to FIG. 9 the registration and authentication processes used by a user to gain access to DMS system 17 are described. At step 170, the registrant accesses the DMS system registration interface, for example, using a web browser to access the DMS system's registration interface URL. Next, at step 171, the registrant selects a DMS service for which he or she wishes to be registered. At step 172, the DMS determines whether the registrant already has an existing DMS service account.

Detailed Description Text (130):
If the registrant already has a DMS account, registration for a new service

requires that the registrant provide his or her user credentials at step 173 and then authenticate those credentials at step 180. If the registrant has no pre-existing account, determined at step 172, the registrant is requested to provide personal information, such as name, address, notification address (e.g., e-mail address, telephone number, IP address), payment information, etc. at step 174. At step 175, the DMS server computer processes and verifies the registration information. If the information is not successfully verified at step 176, the registrant is informed that insufficient information has been provided, at step 186, and the registrant is requested to resubmit the information.

Detailed Description Text (131):
If the information is successfully verified, the registrant is provided with user credentials over a secure link at step 177. User credentials, which may consist, for example, of alphanumeric user IDs, alphanumeric passwords, digital certificates, and/or notification addresses, permit the user to securely access documents, upload documents, view authorized information on documents, digitally sign documents, etc. A user's credentials uniquely identify the user to the DMS system. At step 178, the registrant is given instructions to authenticate his or her credentials.

Detailed Description Text (132):
Once the registrant is issued credentials, or is determined to already have credentials, the authentication process begins, at step 179. This may be accomplished by the registrant accessing the DMS authentication interface by inputting the URL associated with the DMS authentication interface into his or her web browser. Once the registrant is successfully authenticated, at step 180, the registrant's new service account is ready for use at step 181. If the registrant is not successfully authenticated at step 180, an authentication failure is logged at step 182. If the number of authentication failures exceeds a predetermined number, at step 183, the registrant's ability to authenticate is locked for a predetermined period of time at step 184. If the number of authentication failures does not exceed the predetermined safety limit, the registrant is prompted to authenticate again at step 185.

Detailed Description Text (133):
Referring now to FIG. 10, after a user has become registered and has authenticated his or her credentials with the DMS system, the user then may access the services provided by the DMS service by logging on to the DMS system. A user first accesses the DMS logon service at step 190, for example, using a web browser to access the URL associated with the DMS logon service. The user then supplies his or her credentials at step 191, and the DMS checks to see if the credentials are valid at step 192. If the credentials are valid, a user session is created at step 193 and the user is given access to the DMS system at step 194.

Detailed Description Text (136):
HTTP sessions are stateless, so information on these sessions must be maintained in database 25. Communications to server 20 contains a session identifier number that references session information in database 25. Sessions are managed by an automatic process, illustrated in FIG. 11, that continually monitors the length of a session to determine if a current session is longer than a specific, predetermined interval. If there is an active session, determined at step 200, the DMS system determines if the session length is greater than the predetermined interval, at step 201. If the interval has been exceeded, the user session is rendered inactive at step 202 and a flag to that effect is entered in the corresponding database entry. The process is repeated at step 203 for each active session. Alternatively, a user forced logout/exit also may render a user session inactive and the corresponding database entry is flagged accordingly.

Detailed Description Text (138):
Referring now to FIGS. 12A and 12B, the notification request and confirmation

services available on a preferred embodiment of DMS system 17 are described. Notification messages are generated by notification server 35 in response to various user events. For example, when a registrant registers for a DMS service, the registrant receives a notification with instructions on authorization, as discussed hereinabove with respect to step 178 of FIG. 9.

Detailed Description Text (140):
With respect to FIG. 12A, the notification request process performed by DMS system 17 is described. At step 210, a notification message is created by notification server 35 responsive to some user-initiated event. At step 211, a notification request is created that contains some or all of the following information: (1) the subject of the message; (2) the Originator's notification address (e.g., an e-mail address); (3) the notification address of the Authorized User(s); (4) the priority of the notification (e.g., high, medium, or low); (5) the body of the message, including a unique notification identifier created by the DMS system; (6) optionally, an indication of the date and time that the message should be delivered; (7) a status flag (e.g., "pending", "sent", or "failed") indicating the status of the notification delivery, initially set to "pending"; (8) the transport type for the notification (e.g., e-mail, voice message, etc.); and (9) a retry counter that tracks the number of times that a notification request has been processed (initially set to zero, and incremented upon each unsuccessful delivery attempt until the notification request status is marked "failed.") The notification request is queued, at step 212, with a status of "pending," in notification information tables 66 of DMS database 25.

Detailed Description Text (141):
The notification delivery process is described with respect to FIG. 12B. At step 220, the system iterates through the records in the tables with a "pending" flag. At step 221, notification server 35 attempts to deliver the notification using the specified transport system for that Authorized User. DMS system 17 then checks to see if there is a transport rejection, at step 222, for example, if notification server 35 is not working. If no transport rejection is detected, the notification request flag is set to "sent" at step 223, and the notification transaction is logged as sent at step 224.

Detailed Description Text (142):
If a transport rejection is detected, at step 222, a retry counter is checked at step 225. If the number of retries does not exceed a predetermined limit, the retry counter is incremented at step 226 and the notification process begins again. If the number of retries exceeds the predetermined limit, the notification request flag is set to "failed" at step 227, and the notification transaction is logged as "failed," at step 228. At step 229, the DMS system checks information on the origin of the notification request, where the origin of a notification request may be either the DMS system or a system user.

Detailed Description Text (144):
It is possible for a notification to be sent, but for the send to be unsuccessful, for example, if the notification recipient's e-mail address is incorrect. For this reason, each notification transport that delivers notification messages also preferably receives messages that notifications have been sent successfully but have failed during transport. Each notification transport is polled by an automated process for any new messages. Upon receiving a failed notification, this process determines (if possible) the notification identifier, marks the original notification request as "failed" and logs a failed notification transaction linked to the original notification request. In addition, if the origin is not the DMS system, a notification is generated and sent to the sender indicating a failed delivery.

Detailed Description Text (146):
Referring now to FIG. 13, the process by which access tokens are created and used

to control a user's access to the services of the DMS system is described. The present invention uses access tokens as an added security feature to ensure that only validated users access the services of, and documents stored in, the DMS system. As heretofore stated, an access token of the present invention is comprised of a signed string unique to a transaction and generated from one or more random numbers independent of any user information or resource information or any other identifiable information. The following description describes one method by which access tokens of the present invention can be generated.

Detailed Description Text (147):
At step 235, server computer 20 generates two random strings of alphanumeric data, T and K.

Detailed Description Text (148):
At step 237, server computer 20 generates an access token by: (1) concatenating an expiry timestamp for the access token (or a timestamp token from a timestamping authority) (TST) to T resulting in T+TST; (2) hashing K using a well-known hashing algorithm such as MD5 (described in RFC (Request for Comments) 1321) resulting in H (K); (3) using a known symmetric encryption algorithm to encrypt T+TST with H(K) resulting in a message authentication code (MAC); and (4) concatenating T+TST+MAC, where T, TST and MAC are all of known lengths. Server computer 20 then uses the access token to generate a URL:

Detailed Description Text (152):
At step 243, server computer 20 checks the validity of the access token from the URL. Since T, TST and MAC are of known lengths, server computer 20 is able to parse the access token into T' (the portion of the URL that should match T); TST (the portion of the URL that should match TST); and MAC' (the portion of the URL that should match MAC). Security information tables 68 are examined using T' as the primary key. If there is no database record in security information tables 68, the user's request is invalid and the user is not given access to any services offered by the DMS system. If there is a database record, the expiry time of the access token is examined--if the access token is expired then the user is denied access to the DMS system. If the access token is not expired, then TST' is compared to the expiry time of the access token (stored in the database record)--if these do not match then the access token has been tampered with and the user is denied access to services. If there is a match, the next step is to validate MAC' with the information from the database record, i.e., hash K to obtain H(K) and then use H(K) to encrypt T+TST to obtain MAC. Once MAC' has been validated, the access token is validated.

Detailed Description Text (155):
At step 247, server computer 20 determines the user's access rights with respect to the services provided by the DMS system and documents stored in the DMS system by examining security information tables 68, document information tables 61 and user information tables 62 of database 25. At step 249, the user is granted appropriate access rights to the requested DMS system services and to documents stored in the DMS system.

Other Reference Publication (1):
Arnold-moore, Tim et al., "Architecture of a Content Management Server for XML Document Applications", Proceedings of the First International Conference on Web Information Systems Engineering, Jun. 19-21, 2000, vol. 1, pp. 97-108.*

Other Reference Publication (2):
Matyszkiel, Robert , "Implementation Of Universal Document Servers To Transfer Management Information in Tactical Network", IEEE Military Communications Conference Proceedings, MILCOM 1999, vol. 1, Oct. 31-Nov. 3, 1999, pp. 502-505.*

Other Reference Publication (6):

Borenstein, Nathaniel et al., A Multi-media Message System for Andrew, USENIX Winter Conference, Dallas, TX, pp. 37-42 (Feb. 9-12, 1988).

Other Reference Publication (9):
Kent, Jack et al., Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS, Proceedings of the Fourteenth International Conference on Very Large Data Bases, Los Angeles, CA, pp 112-123 (1988).

Other Reference Publication (20):
Overview of the Trans-Virtual Enterpriser Server, Product Overview.

CLAIMS:

1. An Internet-based document management system comprising:

an Internet-based store for storing an electronic document;

a database including a document table for storing information about the electronic document; a user table for storing information about users of the Internet-based document management system; a security information table for storing information about access tokens; and a transaction table that stores information about transactions performed on the electronic document;

a server connected to the Internet-based store and the database, the server programmed to receive the document from a remote computer using an Internet protocol and store the document in the Internet-based store, the server programmed to provide a plurality of services supported by the database and to generate and validate access tokens; and

a notification server connected to the server, the notification server generating and dispatching notifications comprising the access tokens.

7. The Internet-based document management system of claim 1 wherein the server is programmed to filter the electronic document before storing the document in the Internet-based store.

10. The Internet-based document management system of claim 1 wherein the database further comprises an account information table including accounting data, and the server is programmed to apply the accounting data to the information stored in the transaction table to determine a price reflecting usage of the document management system.

11. A method of providing Internet-based document management comprising:

providing an Internet-based store, a database, a server connected to the Internet-based store and the database, and a notification server connected to the server;

accepting a connection from a first remote computer to the server using an Internet protocol;

receiving an uploaded electronic document from the first remote computer to the server using an Internet protocol;

generating a record in a document table of the database to store information about the electronic document;

generating a record in a user table of the database to store information about users of the electronic document;

generating a record in a transaction table of the database to store information

about transactions performed on the electronic document;

generating an access token;

generating a record in a security information table of the database to store information about the access token;

storing the electronic document in the Internet-based store;

accepting a connection from a second remote computer to the server using an Internet protocol;

providing to the second remote computer a plurality of document management services supported by the database; and

providing to the second remote computer a notification comprising the access token.

20. The method of claim 11 wherein the database further comprises an account information table including accounting data, the method further comprising applying the accounting data to the record in the transaction table to determine a price.

L53: Entry 7 of 12 File: USPT Nov 6, 2001

DOCUMENT-IDENTIFIER: US 6314555 B1
TITLE: Software system generation

Abstract Text (1):
A system for building collaborative software agents is provided with a set of editors for capturing data for installation in the individual agents. The collaborative software agents will normally form a community, including some standard agents, provided by the system, and will collaborate to provide functionality in a domain selected by the user. Each collaborative software agent built by the system is provided with co-ordination policies, selected by the user, and represented by a co-ordination graph. A single collaborative software agent can be provided with more than one collaborative policy and is capable of running more than one collaborative policy simultaneously with different agents of the system. An exception handler flags an exception during use of the collaborative agents in the relevant domain when the value of a variable for an agent conflicts with a relevant constraint. Alternatively, the exception handler flags an exception when the resource and time constraints cannot be met by allocation of tasks between the collaborative agents. Communities of software agents built within a system might be used to launch and/or manage telecommunications services or to control a chemical process, for example.

Brief Summary Text (9):
Different types of agent-based systems are described in many papers, such as those published in the proceedings of the First and Second International Conferences on the Practical Application of Intelligent Agents and Multi-Agent Technology. These are published by the Practical Application Company Ltd., Blackpool, Lancashire, in 1996 and 1997 respectively. A general comprehensive review of agent-based technology is given by Hyacinth S. Nwana, "Software Agents: An Overview" in the Knowledge Engineering Review journal, Vol. 11, No. 3, pages 205-244.

Brief Summary Text (35):
Advantageously, the scheduling means may store the task data together with an indicator of status selected from at least two alternative statuses such as "tentative" and "firm". The indicator of status may be used by the scheduler to determine modes of managing such data. For instance, the scheduler may operate a time-out in relation to task data having the status "tentative", after which the data is deleted or can be overwritten by subsequent incoming data.

Detailed Description Text (12):
An event model together with an applications programming interface (API) that allows programmers to monitor changes in the internal state of an agent, and to control its behaviour externally

Detailed Description Text (16):
It further provides support agents of known general type such as name-servers, facilitators (classified directories), and visualisers.

Detailed Description Text (25):
External to the agent system 100, there is a communications system 125 with various components. There is within the external system 125, for instance, a terminal 155,

a software application providing authentication 160 and several network links 165, 170, 175. One of the network links 175 is provided with an external agent 180 which is separate from the agent-based system 100 built using the CABS platform.

Detailed Description Text (27):
In an example, in the system shown, if the user wants data downloaded to the terminal 155, the user agent 110 will have the task of providing an authentication resource 160, the terminal agent 105 will have the task of providing sufficient storage for the data at the terminal 155 and the network agents 115, 120 will have the task of providing network bandwidth to carry the data to the terminal 155. The network agents 115, 120 will also need to collaborate with each other, for instance by bidding against one another in terms of cost, time and quality of service to provide the network bandwidth.

Detailed Description Text (28):
After an inter-agent collaboration stage, the agents 105, 110, 115, 120 will carry out the tasks by outputting control messages to the components of the system 125 they control. The terminal agent 105 must therefore control the terminal 155 so as to provide the storage capacity it has agreed to provide. The user agent 110 must launch the authentication application 160. One of the two network agents 120 must provide the bandwidth, agreed as a result of the collaboration, on its respective network link 170.

Detailed Description Text (29):
During their activities, the agents 105, 110, 115, 120 will have access to common resources within the CABS agent system 100, including for instance the infrastructure agents mentioned above; a name server agent 135, a debugging/fault finding agent 140, referred to here as a visualiser, and a facilitator agent 145.

Detailed Description Text (42):
In use, the agent is driven by events which cause the agent's state to change. The agent will run an event model provided by the component library of the system to monitor these internal changes, making them visible to a programmer via an API. There are three possible external event sources (see FIG. 2): external messages from other agents into the Mailbox 200 (e.g requests for a service), external events initiated from the Execution Monitor 250 monitoring external systems 240 (e.g. from various sensors) and external events initiated from changes to the Resource Database 225. For example, if there is an event which changes the state of the agent, such as loss of a resource, it will need to update its records accordingly.

Detailed Description Text (43):
A change in state may initiate a sequence of activities within and/or outside the particular agent. For example, losing a resource (e.g. through failure) which is required to provide some service would require the Planner & Scheduler module 220 to attempt to secure another resource which may be able to do the same job. If it succeeds, all activities as a result of the loss of the resource can be contained within the agent. However, if the Planning and Scheduling module 220 cannot locate another local resource, the Coordination Engine and Reasoning System 210 will be called upon to attempt either to secure the resource from some other agent or delegate/contract the task which required that resource to another agent. In both cases, the Coordination Engine and Reasoning System 210 will request the Mailbox 200 via the Message Handler 205 to construct a message and despatch it to selected other agents. In this way, coordination of activities with other agents is realised.

Detailed Description Text (46):
The mailbox 200 is implemented as a multi-threaded module with inter-agent communication via TCP/IP sockets. One thread of execution, the server thread, continuously listens for and accepts incoming TCP connection requests from other

agents, receives any incoming messages from those agents and puts the received messages into an in-tray (a queue). A second thread, the client thread, opens TCP connections with other agents to deliver messages. Messages to be delivered are retrieved from an out-tray (queue). When other modules of the agent request a message to be delivered to another agent, those messages are placed on the out-tray of the mailbox to be later picked up by the client-thread. The message language used in the current implementation is the KQML agent communication language [Tim Finin, Yannis Labrou & James Mayfield (1997), KQML as an Agent Communication Language, in Bradshaw, J (Ed.), Software Agents, Cambridge Mass: MIT Press, Chapter 14, pages 291-316]

Detailed Description Text (52):
Agent messages, including inter-agent communications as usually used in co-ordination in CABS, use KQML performatives, and the details of the communications are usually contained in the KQML content field. A CABS agent that puts forward a change in state to other agents (which may represent a proposal, counter-proposal, or acceptance in the CABS co-ordination protocol) uses the KQML performative "achieve". Recipient agents reply by using the KQML performative "reply" when they counter-propose and "accept" when they accept a proposal.

Detailed Description Text (55):
Coordination is extremely important in CABS because its agents are collaborative. Collaborative software agents refer to the complex class of agents which are both autonomous and which are capable of cooperation with other agents in order to perform tasks for the entities they represent. They may have to negotiate in order to reach mutually acceptable agreements with other agents. For instance, an agent may receive a request for a resource from another agent. It will respond according to the relationship between them and according to its own particular circumstances ("state"), such as whether it has that resource available. The response forms part of an interaction process between the two agents which is determined by a "co-ordination protocol". The co-ordination engine and reasoning system 210 allows the agent to interact with other agents using one or more different co-ordination protocols, selected by the developer to be appropriate to the agent's domain.

Detailed Description Text (62):
The overall architecture of the Coordination Software Module 210 is one of a Turing state machine. It takes as inputs various state variable values, parameters, goals, exceptions and constraints, and it outputs decisions.

Detailed Description Text (66):
The co-ordination software module 210 is designed to interpret co-ordination graphs when given an initial (problem) state. The initial state specifies the initial conditions of the problems and the necessary data.

Detailed Description Text (81):
The array describes a graph with four states s1-s4, and five arcs a1-a5. From node s0 we can take arc a1 to state s1 or arc a2 to state s4 or alternative arc a3 to state s4. When more than one can be traversed from a node, the arcs are tried in the order in which they are presented in the graph description.

Detailed Description Text (84):
Referring to FIG. 11, in CABS agents every coordination mechanism is specified in terms of a 14-stage framework where in each stage at least one state process function should be implemented. The 14-stage framework can be considered an "executive summary" of the detailed logic of the co-ordination engine 210 set out in code above. Generic atomic process functions for the fourteen stages are listed below. FIG. 11 describes in schematic form the stages listed below.

Detailed Description Text (144):
exec: to verify whether function is applicable, perform the necessary change of

state, messaging, and certain databases update;

Detailed Description Text (145):
backtrack: to reset all the operations performed and restore the original state;

Detailed Description Text (148):
After a repository of implemented functions has been defined, in order to allow the engine to use a particular coordination graph, a string description of the graph should simply be added to the engine. (The engine can unify multiple coordination graphs into one unified graph which contains no duplication of states.)

Detailed Description Text (262):
In addition, the CABS platform can provide the developer with a suite of support agents 315 of known general type such as name-servers, facilitators (classified directories), and visualisers. Overall, CABS allows the system developer to concentrate on domain-specific analysis without having to expend effort on agent-related issues.

Detailed Description Text (322):
Client-server or master-slave coordination occurs when a master assigns a slave a job to perform. This strategy necessarily requires a superior relationship between the slave and master. The contract-net strategy is when an agent announces a task, requesting bids to perform the task from other agents. It then evaluates the bids and awards the contract to the winning agent(s). This strategy does not rely on any organisational relationship between the participating agents. Limited contract-net is similar to the contract-net strategy except that the announcer selects/restricts the agents to whom bids are broadcast.

Detailed Description Text (344):
The ordering specifies the order in which the preconditions should be tried. In this case it states the planner should attempt to obtain a resource satisfying the resource description ?Link-131 before attempting to obtain a resource satisfying the description ?Link-136.

Detailed Description Text (366):
Deadlock: where agents may be contending for shared resources. An agent may grab a vital shared resource and fail to relinquish it for some reason, perhaps because of some failure for example at the level of the individual agent. But this resource is invaluable to other agents who `hang up` waiting for this resource to be relinquished. Basically, deadlock refers to a state of affairs in which further action between two or more agents is impossible.

Detailed Description Text (373):
All the different tools store different state data. Although no tool is capable of providing a complete analysis of the distributed system, the combination of one tool with another can. Different tools provide or suggest diagnoses. Where the evidence of one tool corroborates that from another tool, the trustworthiness in that diagnosis is increased. There is therefore a greater likelihood that the diagnosis may lead to a successful debugging of the problem. Where the evidence is conflicting, the combination of one tool with another may eliminate a hypothesis or be suggestive of other possible diagnoses.

Detailed Description Text (376):
A Statistics Tool: which captures various statistics on individual agents (e.g. the types and the numbers of messages being exchanged by agents). For example, it answers questions like `How many messages were sent from agent A to agent B during some particular time frame?` or `Plot me the number of messages that Agent A has been receiving over time`, etc. The statistics tool provides pie and bar chart type visualisation of individual agent activities in addition to various graphs. With both the video and the statistics tools, you can select the agents of interest and

study their recorded agents' states more closely. This tool can be used to analyse either a single agent or a society of agents.

Detailed Description Text (378):
A Reports Tool: this provides a GANTT chart type presentation of the overall task which an agent for example may be controlling. Typically, an agent enlists the help and services of many other agents, and this tool allows the designer to choose some particular agent and one task (of possible many tasks) that the agent is controlling. This shows up a GANTT chart of how it has decomposed the task (i.e. what sub-tasks there are), which tasks have been allocated to whom, where the execution of the task has reached, and what the statuses of the different sub-tasks are (e.g. running, failed, suspended or waiting).

Detailed Description Text (380):
The visualiser 140 comprises a software agent having the same general framework as any other of the collaborative agents in a CABS system. It has the capability of pulling data off other agents in the system. It is passive in the sense that it does not negotiate with other agents but it registers itself with the name server 135 and also visualises itself.

Detailed Description Text (384):
Primarily, the visualiser 140 sends request messages to other agents for data which they hold. It queries the name server 135 for agent addresses and then uses those addresses. The message handler 205 of the receiving agent routes the received request message appropriately, depending where the data of interest will be located. Data received by the visualiser 140 from all the other agents is stored, retrieved and processed for appropriate display.

Detailed Description Text (401):
Messages between agents can be colour-coded (for easy visualisation) by type, e.g. all request messages in one colour, or by content, e.g. all messages pertaining to a particular job in one colour. In addition, the tool supports the filtering of messages before display. Messages may be filtered by sender, receiver, type or content. For example, a filter option might state "show only counter-propose messages [type filter] from the set of agents . . . [sender filter] to the set of agents . . . [recipient filter] about job . . . [content filter]". These features facilitate debugging by allowing the user to focus-in on the particular messages of interest. Further, combined with the off-line replay facilities with forward and backward video modes (discussed next), they provide a powerful debugging tool.

Detailed Description Text (407):
Each agent only knows about jobs and subjobs which have been allocated to and/or by it. No single agent holds data for the overall breakdown and the state of all the jobs.

Detailed Description Text (417):
iv) Graph the relationships land states), as shown in FIG. 7.

Detailed Description Text (419):
The reports tool provides a global view of problem solving in a society of agents and is useful both as a debugging and an administrative tool. It allows a user to select a set of agents and request that they report the status of all their jobs to it. Next, the user can select an agent of interest and a job owned by that agent. (An agent owns a job if it is scheduled to perform the job or subpart at a root node in a task decomposition hierarchy for the job.) For the selection of agent and job, the reports tool generates the GANTT chart type graph 700 of FIG. 7 showing the decomposition 560 of the job, the allocation of its constituent subparts to different agents in the community, and the relevant states of the job and subparts. Other attributes of the jobs might also be shown on the chart, such as when each agent is scheduled to perform its part, their costs, the priority assigned to them

by the agents, and the resources they require.

Detailed Description Text (420):
Referring to FIG. 13 and returning to the "MakeComputer" task mentioned above, the GANTT chart 700 might be displayed on screen with selection boxes for selecting the agent and task to look at. As shown, the selection boxes 1305, 1310 have been used to select task "MakeComputer" for agent C 1300. The GANTT chart 700 shows the decomposition of the "MakeComputer" task into MakeTonerCartridge (Agent T) 1315, MakeMonitor (Agent M) 1325, MakeCPU (Agent U) 1320 and MakePrinter (Agent P) 1316. The Task Status Key 1335 shows by colour coding (not visible in FIG. 13) that the MakeTonerCartridge and MakeMonitor tasks 1315, 1325 are running, the MakeCPU task 1320 is completed and the MakePrinter and MakeComputer tasks 1316, 1300 are waiting. A dialogue box 1330 has been brought up to show details of the MakeTonerCartridge task 1315.

Detailed Description Text (422):
As mentioned, the job graph created by the reports tool also shows the current status of each subpart of the job, i.e. either waiting, running, completed or failed. Thus from the graph a user is immediately able to determine the overall status of a job, and if the job fails where exactly it did so--which obviously aids the debugging effort. For easy visualisation, the different states of a job can be colour-coded in the graph.

Detailed Description Text (432):
4. "Co-ordination graph" 1415: visually depicts the dynamic state of execution of the co-ordination engine so as to visualise the progress of the realisation of some goal. Where there are multiple goals the agent is trying to achieve, the user can select a particular one in order to visualise its progress. As each node 1420 of the graph is running, it shows GREEN. If that node fails, it shows RED;

Detailed Description Text (433):
5. "Execution monitor summary" 1425: a diary detailing the tasks the agent has committed itself to performing, and the current status of those tasks (i.e. waiting, running, completed or failed) e.g. the monitor might indicate a task which has failed to start because of inadequate resources (for example, results expected from another agent might never arrive or arrive too late), or it might flag an executing task which has been stopped because it over-ran its scheduled allocation of processor time;

Detailed Description Text (437):
As another example, by looking at the current state of the coordination process for a job (from the graphical depiction of the coordination process), a user might be able to infer, for instance, why the job was rejected by the agent. Since each node of the coordination graph indicates a particular state of the process, the trace of a job on the graph informs the user of most of all there is to know about the coordination of that job. For instance, from a trace, a user might infer that Job-O, say, failed because it required certain resources which the agent did not have; further, the agent could not find any other agent that could provide the resource within the required quantity, time, cost and other constraints.

Detailed Description Text (453):
A browser window for the control tool might for instance allow a user to remotely browse, add, delete and/or modify the task specifications of agents in a society. Such a window might show list boxes for the agents in the society, the set of tasks of the current selected agent, and the status of the selected task. A text area might show the specification of the current selected task. "Modify" and "Add" buttons might launch a task specification editor through which an existing task specification may be modified or a new task defined. The status of the tasks are set to OK when, as far as the tool is aware, the task definition in the tool is the same as that in the relevant agent's task database. A staus value of UNKNOWN

indicates that the user has modified the task specification but, so far, the agent has not yet returned acknowledgement of receipt of the new specification.

Detailed Description Paragraph Table (2):
String[ ] arcs // the list of arcs from this mode String[ ] nodes // the list of node pointed to by the arcs Node previous_node // the previous node in the chain Graph graph // the graph which led to the instantiation of this node int state // the current state of this node int current_arc // the current arc awaiting execution Object data // data describing the current state on which this node acts void run(Engine engine) { switch(state) { case READY: if ( !graph.allow_exec ( ) ) // the graph.allow_exec( )asks the graph if execution of this node is allowed fail(engine,false,"Exec refused by graph"); else switch( exec( ) ){ case OK: state: = RUNNING; engine.add(this); break; case WAIT: engine.waitForMsg(this); break; case FAIL: fail(engine,false,"Node exec failed"); break; } break; case RUNNING: if ( ! graph.allow_exec( ) ) fail(engine,true,"Exec refused by graph"); else if ( arcs == null ) done(engine,"terminal node reached"); else if ( current_arc >= arcs.length ) fail(engine,true,"All arcs traversed"); else exec_arc(engine, data); break; } } void fail(Engine engine, boolean backtrack, String reason) { state = FAILED; if ( backtrack ) backtrack( ); graph.failed(engine,this); if ( previous_node != null ) previous_node.nextArc(engine); } protected int exec( ) { // contains the actual executable code for // this node } protected void backtrack( ) { // reset any state changed by exec( ) } void exec_arc(Engine engine) { load process described by current_arc and the call the run( )method of the arc. If the run( )method succeeds { create a new node process by calling graph.newNode(label) where label is the node label pointed to by nodes[current_arc ]. Initialise the new node with setInput (data) and add the node to the engine with engine.add(newNode) } if the arc process cannot be created or the run method fails call the nextArc( )method of this node } void setInput(Object data) { this.data = data } void nextArc(Engine engine) { if ( !graph.allow_exec( ) ) fail(engine,true,"Next arc disallowed by graph"); else { if ( state == DONE && arcs == null ) fail(engine,true,"All arcs traversed"); else { state = RUNNING; current_arc++; engine.add(this); } } }

Detailed Description Paragraph Table (3):
String[ ][ ] nodes // two dimensional array of listing the nodes and arcs of the graph String start_node // the label of the start node String next_node // the label of the next node Node previous_node // the process of the last node of the graph Node begin_node // the process identifier of the start node Graph parent_graph // the graph of which this is a subgraph int state // the current state of the graph public void run(Engine engine, Graph parent_graph, Node previous_node, String next_node) { this.parent_graph = parent_graph; this.previous_node = previous_node; this.next_node = next_node; start (engine,arc_data); } public void run(Engine engine, Object input) { start (engine,input); } protected void start(Engine engine, Object input) { state = RUNNING; begin_node = newNode(start_node); if ( begin_node == null ) fail (engine,"Start node not found"); else { begin_node.setInput(input,previous_node); engine.add(begin_node); } } void done(Engine engine, Node node) { state = DONE; if ( graph != null ) { Node next = graph.newNode(next_node); if ( next == null) { state = RUNNING; node.nextArc(engine); } else { Object data = node.getData( ); next.setInput(data,node); engine.add(next); } } } void failed(Engine engine, Node node) { if (node == begin_node ) state = FAILED; } void fail(Engine engine, String reason) { state = FAILED; if ( previous_node != null ) previous_node.nextArc (engine); } Node newNode(String name) { create a new node process identifed by the label name and using the 2D nodes array find the arcs leaving this node and their destination nodes and set these as the arc/node data for the new node }

Detailed Description Paragraph Table (13):
Society Tool Messages What is Message Type To WHOM Purpose returned Request Agent name_server return addresses Addresses of all agents Request All agents return relationships Relation- ships Request None, some or all cc all messages cc'ed agents messages

h e b b g e e e f c e c e ge

Other Reference Publication (1):
Erman et al., "ABE: An Environment for Engineering Intelligent Systems," IEEE Transactions on Software Engineering, vol. 14, Issue 12, Dec. 1998, pp. 1758-1770.*

Other Reference Publication (4):
Finin et al., "KQML as an Agent Communication Language," Proceedings., Third International Conference on Information and Knowledge Management (CIKM'94), ACM Press, Nov. 1994, pp. 1-9.*

CLAIMS:

8. Software building apparatus as in claim 7 wherein the graph description describes a set of states, identified by nodes, and a set for arcs for traversing between specified states, each node and each arc identifying at least one process.

17. A software system as in claim 16 wherein the graph description describes a set of states, identified by nodes, and a set for arcs for traversing between specified states, each node and each arc identifying at least one process.

28. A method as in claim 27 wherein the graph description describes a set of states, identified by nodes, and a set for arcs for traversing between specified states, each node and each are arc identifying at least one process.

37. A method as in claim 36 wherein the graph description describes a set of states, identified by nodes, and a set for arcs for traversing between specified states, each node and each arc identifying at least one process.

L53: Entry 9 of 12 File: USPT Feb 27, 2001

DOCUMENT-IDENTIFIER: US 6195685 B1
TITLE: Flexible event sharing, batching, and state consistency mechanisms for interactive applications

Abstract Text (1):
A system, method and computer program storage device providing event and/or state sharing support e.g., for building object-oriented interactive groupware in wide-area distributed environments (such as the Internet). For collaborative applications programmed using events, mechanisms are provided for sharing application-specific events. For example: an event based programming model allows applications to post an event and triggers corresponding ERUs (Event Reaction Unit) in reaction to a received event; preconditions for control activation of ERUs; and event consistency policy objects implement application specified event consistency model. Some policy examples are: a policy in which event order is not guaranteed, but all events are guaranteed to be sent to the ERUs eventually; and a policy that first triggers local ERUs and then posts the event to the server. An out-of-order event is detected using the event notification from the server; and an automatic detection of out-of-order events seen by ERUs in the local workstation in this optimistic event execution model. For applications requiring support for state sharing, an asynchronous model for updating replicated state, which supports atomicity of updates across multiple shared objects is described. Coupled with a flexible marshaling framework, this allows existing application data-structure classes to be easily extended and made shareable, while providing support for context-sensitive state marshaling applications. To solve the problem of replica consistency, a novel combination of three mechanisms is used: global locks; detection of incorrect update ordering; and cloning a subset of the shared objects for state re-initialization. To reduce network load due to fine-grained user interaction, a framework for application specified event batching, called Late Event Modification (LEM), enhances the event interface to allow applications to modify the event objects after posting them to the set.

Parent Case Text (2):
The present invention is related to patent application entitled "Flexible State Sharing and Consistency Mechanisms for Interactive Applications," by Mukherjee et al., filed of even date herewith, IBM U.S. Pat. No. 6,058,416. This application, which is commonly assigned with the present invention to the International Business Machines Corporation, Armonk, N.Y., is hereby incorporated by reference in its entirety into the present application.

Brief Summary Text (2):
This invention relates to distributed applications and more specifically to a system and method for providing event and state sharing support for building object-oriented interactive collaborative applications in wide-area distributed environments such as an intranet or the Internet.

Brief Summary Text (4):
With the ability to extend web browser functionality using JAVA (a trademark of Sun Microsystems), plugins, ACTIVEX (a trademark of MicroSoft), etc., simple interactive groupware applications are being made available to a wide population of users. Examples include Internet chat and simple white boards. However, for

building complex object-oriented collaborative applications, developers need a simple and powerful programming model, which at the same time should allow good response time and efficient implementation in a wide-area distributed environment. Appropriate support for sharing state is critical for supporting collaborative applications. In wide-area environments, replication of state is used to improve response time. Also, there are instances where a more primitive mechanism like event notification (also called event sharing) is a much better match for application requirements than state sharing. These high-level requirements for interactive groupware, coupled with replicated state, lead to three critical objectives that are addressed by the present invention.

Brief Summary Text (6):
1. Programming model: One objective, from a system design viewpoint is to decouple the programming model from any concurrency control implementation (including those of the present invention). For example, when an application issues an operation on a shared object, the system should have flexibility in scheduling this operation at the various replicas. This separation between the issuing of an operation and its execution leads to an asynchronous programming model that allows the system to employ a variety of concurrency control implementations such as pessimistic locking, ordering actions via a server, optimistic notification with automatic rollback and others. From the application developers viewpoint, the following requirements are considered:

Brief Summary Text (10):
The present invention has features that provide an asynchronous model for specifying atomic operations on the shared state using update events. This model only requires shared objects and events to implement a simple Marshalable interface. This interface allows for powerful state sharing semantics which are not possible to implement with simpler interfaces like JAVA.TM. Object Serialization.

Brief Summary Text (11):
2. Consistency of shared state: Potential inconsistencies in the replicas can arise due to different ordering of the events at different processes. Most systems take two extreme approaches when dealing with consistency of replicated state, (1) a fully optimistic approach with rollbacks and reexecution, and (2) a pessimistic approach utilizing locking. The first approach frees the application programmer from the burden of consistency maintenance, but the effect of jitters in the user interface due to automatic rollback could be a problem. The second approach does not allow enough freedom of interaction. It has been observed that strict locking is usually not necessary for collaborative applications because of implicit social protocols employed by collaborating users.

Brief Summary Text (16):
3. Application specific event batching: Fine-grained user interaction with a GUI leads to a lot of updates on the shared state. For efficiency reasons, all these updates should not be propagated to the other users in the collaboration. The present invention uses a novel event batching technique (Late Event Modification) to resolve this tension between interactivity and performance.

Brief Summary Text (18):
The foregoing and other objectives are realized by the present invention, which provides a system and method for event and state sharing support for building object-oriented interactive applications in wide-area distributed environments (such as an intranet or the Internet).

Brief Summary Text (19):
One embodiment of the present invention is implemented as middleware that provides support for different classes of distributed collaborative applications over the Internet. It provides an extensible set of mechanisms for event sharing and/or state sharing among a group of distributed objects to suit the requirements of

individual applications.

Brief Summary Text (20):
The system can support both symmetrical and asymmetrical collaborative applications. A symmetrical collaborative application is one which the sets of collaborative objects in each location are identical in their choice of consistency and transport options for sharing of events and application state, whereas an asymmetrical collaborative application may have different consistency and transport options.

Brief Summary Text (27):
a policy that first triggers local ERUs and then posts the event to the server. An out-of-order event is detected using the event notification from the server.

Brief Summary Text (30):
In yet another embodiment for applications requiring support for shared state and an application specified state resolution policy, the invention provides a conflict detection mechanism for a shared set of objects. For example:

Brief Summary Text (31):
an asynchronous model for specifying atomic updates to the shared state.

Brief Summary Text (37):
In yet another embodiment, for applications requiring support for shared state and an automatic state resolution policy, it provides a novel "cloning and re-initialization" mechanism. For example:

Brief Summary Text (41):
cloned subset can be exported to other clients.

Brief Summary Text (43):
In still another embodiment, for application requiring fine-grained user interaction with a GUI (which leads to a lot of updates on the shared state), a novel event batching framework (LEM) is provided to resolve the tension between interactivity and performance. For example:

Brief Summary Text (46):
In yet another embodiment, for applications requiring context sensitive marshaling of the state of its objects, a lightweight yet powerful marshaling framework is provided.

Drawing Description Text (5):
FIG. 3 depicts an example of the Team Server (of FIG. 2) and a logic flow for creating a collaboration team;

Drawing Description Text (6):
FIG. 4 depicts an example of the layers of the collaborative client middleware of FIG. 1;

Drawing Description Text (9):
FIG. 7 depicts an example of a Distributed Shared Events infrastructure at the client side;

Drawing Description Text (11):
FIG. 9 depicts an example of marshaling object state;

Detailed Description Text (2):
FIG. 1 depicts an example of a system having features of the present invention. As depicted, the system includes a local client site 100, one or more remote client sites 170 and a server 120, which are connected using a network 113. The network is

used to communicate messages between clients and the server using a network specific protocol. For example, when the Internet is used as the network, the TCP/IP protocol is used for communication. For clarity, only the relevant components resident in the memory of the clients 100, 170 and server 120 are shown.

Detailed Description Text (3):
The server 120 (either a client machine can be used to run the server 120, or a dedicated server machine can be used) maintains a set of collaborative activities 150 (described in more detail in FIG. 2), one for each active collaboration. The server 120 identifies clients in a collaboration, for example, using their Internet addresses. The server 120 may maintain relevant information about a client such as geographical location (used for efficient multicast protocols), the number of connections to the server 120 and the collaborations that a client 100, 170 is participating in.

Detailed Description Text (4):
Each local client site 100 includes an operating system layer 101, a middleware layer 102, and an application layer 103. The operating system layer 101 can be any available computer operating system such as AIX, MVS (trademarks of IBM) WINDOWS 95, WINDOWS NT (trademarks of MicroSoft), SUN OS, SOLARIS, and JAVA OS (trademarks of Sun Microsystems).

Detailed Description Text (5):
The client includes one or more distributable building blocks ("Components") 105 of a collaborative application 103, wherein each Component 105 may be executed independently and collaborate with other Components. Each Component includes a set of one or more Event Reaction Units (ERU), wherein each ERU has an associated precondition and a reaction. The precondition specifies one or more reference events that trigger the reaction, which may perform any computation or task and may produce other events.

Detailed Description Text (6):
A preferred embodiment of the present invention includes features implemented as software tangibly embodied on a computer program product or program storage device for execution on a processor (not shown) provided with client 100, 170 and server 120. For example, software implemented in a popular object-oriented computer executable code such as JAVA provides portability across different platforms. Those skilled in the art will appreciate that other procedure-oriented and object-oriented (OO) programming environments, such as C.sup.++ and Smalltalk can also be employed.

Detailed Description Text (7):
Those skilled in the art will also appreciate that methods of the present invention may be implemented as software for execution on a computer or other processor-based device. The software may be embodied on a magnetic, electrical, optical, or other persistent program and/or data storage device, including but not limited to: magnetic disks, DASD, bubble memory; tape; optical disks such as CD-ROMs; and other persistent (also called nonvolatile) storage devices such as core, ROM, PROM, flash memory, or battery backed RAM. Those skilled in the art will appreciate that within the spirit and scope of the present invention, one or more of the components instantiated in the memory of the clients 100, 170 or server 120 could be accessed and maintained directly via disk (not shown), the network 113, another server, or could be distributed across a plurality of servers.

Detailed Description Text (9):
The middleware layer 102 implements domain specific system infrastructures on which applications can be developed. As is conventional, the middleware 110 is communicatively coupled to the application and to the network, and is adapted to communicate to and receive messages from one or more remote clients over the

network 113. The Collaborative Client ("collab client") 110, the client side of the system (described in more detail in FIG. 4), preferably belongs to the middleware layer. The collab client 110 receives asynchronous distributed shared events from the network and delivers a received event to an associated Component 105.

Detailed Description Text (11):
The system described in this invention can be implemented on a variety of network platforms 113. By way of example only, a local area network (LAN) setup could be constructed using SUN ULTRA 1 machines connected by fast Ethernet. Similarly, another local area network setup could be constructed using IBM RS/6000 (running AIX OS), and Personal Computers (running WINDOWS 95 OS) as clients 100, 170 connected to a high end RS/6000 machine as the server 120 using a 16 MB token ring network. An example of a wide area network uses SUN ULTRA 1 machines as clients 100, 170 and a SUN SPARCSTATION 20 as the server 120, connected via the Internet.

Detailed Description Text (13):
An active collaboration in the system is encapsulated in a collab activity 150. Each activity has participants (also called clients). When a collaboration is created, a new collab activity 150 is instantiated and bound into a name space of the collaboration. This allows others to also have access to that activity, thus allowing them to join the collaboration.

Detailed Description Text (14):
FIG. 2 depicts an example of the components of an activity 150. Depending on the type of collaboration (or application), an activity 150 can be instantiated with different implementations of these components. Thus, activities are highly configurable. A Door component 200 implements a named communication port which can be used by users to connect to an activity. For example, when a TCP/IP protocol is used over the Internet for communication, the Door 200 is realized using a named TCP/IP port [1]. For example, the door can be implemented by a method waitForClient ( ) in a predefined interface that waits for incoming client connections 205. When no implementation of Door 200 is specified, a default implementation using the TCP/IP protocol can be used.

Detailed Description Text (15):
A new thread 280 is instantiated to monitor the incoming client connections 205 to an activity 150. Once a new client connection is received, the client can be authenticated using an Auth object 202. An Auth object implements an application specific authentication policy. Depending on its authentication requirements, an application 103 can implement an Auth object 202 by implementing an authenticate( ) method of a predefined interface. The most commonly used Auth objects (e.g., password authentication, and no-authentication) are provided by the system presented in this invention. Once a client is authenticated, a connection to the client is established, a client object 240 is created inside the activity, and a new thread from a pool of threads 241 is assigned to receive client requests. Again, depending on the protocol being used, an activity 150 uses an appropriate connection class to communicate with connected clients. For example, when the TCP/IP protocol is used, a connection class uses socket APIs to communicate with the clients. A conventional data structure called the client connection table 220 is used to hold the connections to active clients.

Detailed Description Text (16):
Each activity 150 maintains a database 207 of the clients currently participating in the collaboration. This database 207 allows queries on client information. The clients participating in a collaboration can be grouped in overlapping collaboration teams 230 and sub-teams. A team server 205 in an activity maintains a number of named teams. Teams can be dynamically created and destroyed and a client can join or leave a team.

Detailed Description Text (17):

Each activity also maintains a method table 210 including a set of registered methods 250 to provide application specific services. These services may include queries and updates of user databases 207, creation and update of collaboration teams and others. An activity 150 will need application specific implementation of these services. For example, unlike a public meeting, an anonymous meeting does not allow a client to query the names of all the participants. This can be achieved by registering the appropriate implementation of a method getUsersInTeam( ) with an activity.

Detailed Description Text (18):
A session manager 160 (FIG. 1) [2], which itself is a collaborative application, can be used to manage a set of concurrent activities and their participants. The way the session manager structures the activities and the resources in an activity, determines the metaphor of a collaboration server. The use of such metaphors are well known in the conferencing and collaboration arts (see e.g., H. Chiu et al., "Conferencing Metaphor," IBM TDB, V. 36, No. 32, (February 1993); and U.S. Pat. No. 5,634,129, issued May 27, 1997 to Dickinson et. al., entitled Object Oriented System for Representing Physical Locations, which are hereby incorporated herein by reference in their entirety). For example, a "room metaphor" (the default metaphor in one embodiment of the system), organizes the collaborative activities in separate rooms. A Drawing room contains collaborative drawing tools and a color printer, whereas an Auction room will contain ongoing auction activities. Similarly, a conference center metaphor will implement only one room containing at least two applications: "reserve a conference" and "join a conference". The system preferably uses a hierarchical naming scheme, similar to URLs, to name activities. For example, the activity 150 can be named using a server name and a room name (in case of the room metaphor). The teams 230 and sub-teams in an activity 150 can be similarly named.

Detailed Description Text (21):
FIG. 3 depicts a more detailed example of the Team Server 205 and a logic flow for creating a collaboration team. As depicted, when a collab client 110 sends a request to the Team Server 205 to create a Team 230 of a certain type (the createteam method call 305), the team server uses its policy factory object 320 (the factory object knows how to create a policy of a certain type) to create a team policy object 330 for the specified team type. Once a team policy is instantiated and a team object is created, they are inserted into a policy table 340 which maps the created team to its policy object.

Detailed Description Text (22):
FIG. 5 depicts an example of the interface of the Team Policy 330. As depicted, a TeamPolicy object 330 defines the type of a team 230. A team can be instantiated either with an existing type or a new application-defined type. A certain behavior of a team 230 can be programmed by implementing the following functions of the TeamPolicy object. For example, a team can implement an authentication policy for its members using the addNewMember 510 function. Similarly, a required performance for event distribution can be achieved by implementing a suitable multicast 520 and unicast 525 protocol. By way of example only, the following methods can be included in one embodiment of the present invention:

Detailed Description Text (23):
1. addNewMember( ) 510: a function defining the behavior of a team when a new client joins the team.

Detailed Description Text (24):
2. deleteMember( ) 515: defines the policy of a team when a client leaves the team.

Detailed Description Text (29):
FIG. 4 depicts an example of the collab client 110 having features of the present

invention. As depicted, the client side middleware 102 layer includes several layers. In an Internet embodiment, the TCP/IP and Group communication protocol layer [1] [4] is preferably used to implement a distributed shared events layer 420. The Distributed Shared Events (DSE) layer 420, provides support for sharing application specific events among application components residing in client machines 100, 170 distributed in a wide area network such as an intranet or the Internet. Finally, support for shared objects 190 and state consistency 430 is provided to the applications by the Distributed Shared Objects (DSO) layer 430. Unlike existing systems supporting distributed shared state, the DSO layer 430 decouples conflict detection 431 and conflict resolution 432 by providing two levels of support to application components 105. The conflict detection level 431 detects conflicts of updates to shared state (described in detail in FIGS. 13-16). At a first level of support, once the conflict is detected, the application components 105 are informed. An application may implement its own conflict resolution policy using this information. At a second level of support, the DSO layer 430 provides application components 105 with a unique conflict resolution policy (described in detail in FIG. 11) which is suitable for collaborative applications. A collaborative application 103 can be programmed using the DSE support 420 for shared events, the DSO 430 support for shared state, or both. Both the DSE layer (FIGS. 6, 7) and DSO layers (FIGS. 6, 8-16) will be discussed in more detail below. Those skilled in the art will appreciate that many of the features of the DSE and DSO (middleware) layers could be alternatively implemented in the application layer 103 within the spirit and scope of the present invention.

Detailed Description Text (31):
FIG. 6 depicts a more detailed example of the Application Components 105 of FIG. 1. As depicted, a Component 105 is a distributable building block of an application 103. Each Component may be executed independently and communicates with other Components as necessary. A Component can be created using the DSE and DSO support provided by the system for sharing events and state, together with application-specific code. The specific behavior of a Component 105 is programmed with Event Reaction Units (ERU) 630 and shared object sets 680 (described in more detail later).

Detailed Description Text (32):
Applications developed using shared events can be programmed with ERUs. According to the present invention there are two main parts to the specification of an ERU: the condition (also called precondition or trigger), and the reaction. The reaction is the body of the ERU and is executed when the condition is true. When an application component 105 is initialized, it registers its ERUs 630 with the collab client 110. An example of a registration mechanism is a table where each record includes an ERU identifier and its preconditions. As asynchronous events arrive, the collab client 110 checks if any of the registered ERU's condition is `true`. Once the condition is satisfied, the collab client 110 delivers 610 the event to the Component. The Component 105 then reacts to the event by executing the reaction of the corresponding ERU 630, which can be synchronous (as a function call) or asynchronous (as an independent thread).

Detailed Description Text (33):
The implementor of an ERU 630 can control its activation by associating a condition routine with that ERU. The role of such a routine is to define a scheduling policy for an ERU's reaction. A condition is defined as the satisfaction of requirements on the arrival of events 610. Therefore, associated with each ERU is a conditional function which, given a set of reference events and a set of received events, determines whether some condition for executing the body of the ERU has been met. For example, in a collaborative foil flipper supporting annotation, the ERU 630 responsible for displaying annotations on a foil does not get triggered until both an annotation event and a foil flip event (to flip to the foil which needs to be annotated) are received at a client. This can be achieved using a conventional "AND" logic condition which returns true if all reference events required by the

ERU are received.

Detailed Description Text (37):
FIG. 7 depicts an example of several data structures and a logic flow for sharing events. As depicted, when a client 100, 170 joins a named team 230, a handler object (TeamDescriptor object) 725 is instantiated at the client. The Team Descriptor 725 could be any conventional handler object which provides an interface to a client, that the client uses to communicate with the team 230. All the client objects in a team can post events 751 to the team members (possibly via the server 120). Each team 230 preferably has a policy 330 on how events are distributed to its members. Depending on the event distribution policy required by a team, the TeamDescriptor object 725 implements a policy in the client 100, 170. This policy is responsible for ordering of events posted 751 in a team (ranging from ordering everything through the server 752 to dispatching locally generated events 753 immediately), and dispatching of received events. For example, a simple event distribution policy dispatches received events to pre-specified objects. An enhanced version dispatches events using the target field of the java.awt.Event [3] object, and does automatic translation of the target to and from a machine-independent reference. This allows single user applications to be made collaborative by establishing equivalence between certain user-interface (UI) objects and then broadcasting user events to everyone. As described below, a complex policy can also be used to provide efficient support for shared state for collaborative applications.

Detailed Description Text (38):
The server team object 230, after receiving an event, can multicast it to all the clients (100, 170) in the team, or can unicast 525 the event to a particular client 520 using the underlying network communication protocol. A client preferably can query the list of available teams 230 in the server 120, and can also query the members of each team. A client joins a team in three roles (1) sender, (2) receiver or (3) both.

Detailed Description Text (39):
Referring again to FIG. 7, the example DSE mechanism includes a dedicated thread called the Client Communication Thread (CCThread) 710 in the client waiting for messages. Messages are preferably used for communication between the collab client 110 and the collaborative activities 150. In step 701, a message arrives from the server 120. A message may either contain an event posted to a team or it may carry information required by a collab client 110 (called infrastructure messages). On receiving the message, the thread 710, checks if the message is for the collab client infrastructure 715 or a TeamDescriptor object 725, in step 702. Infrastructure messages 740 are dispatched to the infrastructure 715. If the message is targeted for a team descriptor 725, it looks up the descriptor for which it is intended using a TeamDescriptor Lookup Table 720. In step 703, the message is then dispatched to the appropriate TeamDescriptor 725. If using a Simple event distribution type, this message potentially contains an event 620 posted to a team by an application component 105. In this case, the event is placed in a Shared Event Queue 730, in step 704. A Shared Event Thread 735 is then responsible for picking up events from the queue and dispatching them to the registered ERUs 630, in step 705. The dashed arrows demonstrate how a TeamDescriptor could implement a (752) "post event" interface (used by a client to post an event to a team) for an application. On receiving the post event 751, the TeamDescriptor 725 sends it (in step 752) to the server as well as directly puts it (in step 753) in the local shared event queue 730, to avoid the round-trip delay from the server.

Detailed Description Text (41):
Referring again to FIG. 6, a set abstraction is preferably used to represent a collection of related shared objects 680. Objects 650 in a shared set 680 can have pointers to other objects in the same set, which can be cyclic (see FIG. 8). Applications in a collaboration can dynamically join or leave a set. All the

objects 650 in a set 680 are fully replicated at all the applications 103 that have joined that set of objects. This asynchronous model allows the specification of atomic operations on the shared state using update events. The three main operations on an object set 680 are:

Detailed Description Text (45):
FIG. 8 depicts an example of an event object 810 and a shared set 820 containing three objects: an object 830 of type Name; and two objects of type Account 840, 850. The event object 810 includes a handleEvent method to update the state of the two objects 840 and 850 in the shared object set 820 (it subtracts $500 from one account object 850 and adds the amount to the other account object 840).

Detailed Description Text (46):
The addObject method: adds an object o to a local set replica 680; sends a message with the new object state to other processes; and then returns. In contrast, the other two operations are asynchronous, i.e., they are executed locally and then remotely at some time in the future.

Detailed Description Text (48):
To propagate the state of the newly added objects or the update events to other processes in the collaboration, the state needs to be marshaled. Most systems assume that an object knows how to marshal itself, and therefore expect it to implement a simple marshal method to which they pass an output stream as a parameter [3]. However, for complex applications, the marshaling of an object is very context dependent. For example, let `A` be an event object being added to a shared object set, which has a pointer to `B`, (an object already in the set). The object `A` needs to be marshaled, as its state has to be sent to the other replicas. But when marshaling object A, object B is not marshaled, and a machine independent handle is substituted for it. However, when a new client joins the set, the state of both A and B have to be marshaled, so that the newcomer can be initialized. More complex scenarios arise when the system needs to provide support for object cloning and state consistency. However, the basic inference is that the system needs to provide a context for the marshaling, which the present invention does by using a MarshalHelper object.

Detailed Description Text (51):
/* out: output stream to write the state */

Detailed Description Text (53):
/* in: input stream to read the state */

Detailed Description Text (69):
FIG. 9 depicts an example implementation of a shared conventional linked list 910 using shared sets. An update event can be used to append another list 920 to the end of this shared list 910. As discussed, when marshaling the queue field 922, the MarshalHelper substitutes a machine independent handle, while it marshals the complete state of the append field 923. This flexibility cannot be achieved with a marshaling interface like JAVA object Externalization [3], where the object being marshaled needs to keep some information about its context and use it in the Marshalable implementation. This interface is also especially useful when doing partial checkpointing.

Detailed Description Text (71):
State Consistency

Detailed Description Text (72):
Potential inconsistencies in the replicas can arise due to different ordering of the events at different processes. Most systems take one of two extreme approaches when dealing with consistency of replicated state: (1) a fully optimistic approach with rollbacks and reexecution; and (2) a pessimistic approach utilizing locking.

The optimistic approach frees the application programmer from the burden of consistency maintenance, but the effect of jitters in the user interface due to automatic rollback could be a problem. The pessimistic approach has the problem of not allowing enough freedom of interaction. It has been observed that strict locking is usually not necessary for collaborative applications because of implicit social protocols employed by collaborating users. The present invention uses an intermediate approach which uses optimistic execution along with a combination of three mechanisms which flexibly expose the application writer to more of the distributed nature of the application.

Detailed Description Text (73):
This section discusses the three level approach to state consistency. First, global locks are provided to guarantee correct ordering of events at all clients. When a client updates the state of a shared object: it first acquires a global lock; issues the update events; and finally, releases the lock. Since the lock release is ordered after the events issued while the lock was held, updates to a shared set are completely ordered. Second, for applications in which locking is not always desirable, the system also provides conflict detection. Third, a cloning and re-initialization method can be used to construct an application specific conflict resolution policy.

Detailed Description Text (75):
The conflict detection algorithm depends on the architecture of the system. By way of example only, the present invention assumes an architecture where clients (100, 170) are logically arranged in a star configuration with a server 120 in the center. All events flow through the server, and the order in which the server receives the events is the `correct` order. Events originating at a client 100 are immediately executed locally and also sent to the server 120. This optimistic execution can result in inconsistency. The server 120 distributes the events to all the clients (100, 170), including the source, using conventional FIFO communication channels.

Detailed Description Text (76):
Let X=O.sup.1, O .sup.2, . . . , O.sup.m be a current set of shared objects 680 (FIG. 6). An event e issued on this set has a read-set e.R.OR right.X and a write-set e.W.OR right.X. Each client process P maintains an "unstable" queue of events. These are the events which originated at P and have been executed at a local client 100, but have not yet been reflected back by the server 120. Let a remotely originating event e" be received at P. The possible reasons for conflict are:

Detailed Description Text (80):
FIGS. 13-16 depict examples of a logic flow corresponding to the following pseudo-code process, which describes a simplified method for maintaining a set of objects 680 whose state is in error at a client process P. These objects are in the error set. For each object o, the fields o.numOfR and o.numOfW give a count of the number of times o is read and written respectively by events in the unstable queue. A localEventReceived is a method called when an event is generated locally and localEventReflected is called when such an event comes back from the server (FIG. 14). When a remote event is received, the method uses the read and write set information of that event and the events in the unstable queue (which should have been ordered later), to detect errors (FIGS. 15, 16).

Detailed Description Text (115):
As depicted in FIG. 13, when an event is received at a client (100, 170) from the server 120, in step 1310 in step 1320 the collab client 110 checks, if the event was generated by a local application component 105. If yes, in step 1330, (as depicted in FIG. 14), the collab client 110 removes the event from the "unstable queue" 1410. Then, for every object in the read-set of the received event, it decrements a read counter of the object, in steps 1415, 1420, and 1430. Similarly, for each object in the write set of the event, it decrements the write counter of

the object, in steps 1425, 1440, and 1450.

Detailed Description Text (116):
Referring again to FIG. 13, if in step 1320 it is determined the event was generated by a remote application component the process proceeds to step 1340, (as depicted in FIG. 15). Referring now to FIG. 15, the collab client 110 checks if the write counter of any object in the write-set of the event is positive, in steps 1510, 1520, and 1530. If found positive, it marks the object as in error, in step 1540. Otherwise, it checks if the read-counter of the object is positive, in step 1550.

Detailed Description Text (117):
If positive, it marks the object as in error, in step 1560. Similarly, the collab client 110 checks if any object in the read-set of the event has a positive write counter, in steps 1570, 1535, and 1525. If yes, it marks all the objects in the write set as in error, in step 1515. Finally as depicted in FIG. 16, for each event in the unstable queue it checks if any object in the read-set of the event is in error, in steps 1610, 1620, and 1630. If yes, it marks all the objects in the write-set of the event as in error.

Detailed Description Text (121):
FIG. 10 shows an example of the effect of cloning a subset 1001 of objects: a 1011; b 1013; and c 1012, in a shared set 1000. The created clone of the subset 1001' includes objects: a' 1033; b' 1032; and c' 1031. As depicted, the clone object a' 1033 preserves the pointer (1040, 1040') to an outside object d 1022. The pointer 1050 from the object e 1021 to the original object b 1013 is not modified. The cloned subset 1001' can be added to the original shared set of objects (as soon an object or a set of objects is added in a shared set, the added objects become shared). This provides a mechanism to export incorrect state to other clients, so that different versions of the same object can be compared. It can then be used to re-initialize the state of the originals from the clone.

Detailed Description Text (122):
FIG. 11 depicts an example of how the cloning and re-initialization mechanisms can be used to correct diverging state. The collab client 110 detects (using the conflict detection algorithm described above in FIGS. 13-16) that the replicas of an object B (1105, 1105') of an application component 105 at a local client 100 and remote client 170 are different and informs the application component 105. The components clone (in step 1115) both A 1110 and B, (since meaningfully interpreting B alone is not possible). These clones (1120, 1120') are exported (in step 1125) to each other by adding them to the corresponding set of shared objects. Now that the clients have local copies of the differing states, they can compare and select one of these to represent the correct state. In the example, they select object B". In step 1140, the object B" 1130 is then used to re-initialize the originals by client 100.

Detailed Description Text (124):
The re-initialization can be done using a special type of update event, called a re-initialization event. This event does not read any objects, but re-initializes the state of all the objects in its write set. The following pseudo-code logic depicts an example of how this event can be processed by the collab client 110. The processing for a local re-initialization event reflected back from the server remains the same.

Detailed Description Text (138):
In a preferred programming model, shared objects 650 are modified only through update events issued using the updateEvent method. The system executes these events locally as well as multicasts them to all the processes in the set, which then execute the events to bring about the same state change. To get good response time for the issuer's actions, these events should be fine-grained. For example, in a

text editor, every keystroke may have to be packaged as an event so that an issuer (say Alice) sees the result of her actions immediately. But in most cases it is not necessary for other users to see the keystroke generated by Alice immediately. In fact, Alice may not want them to see her modifications until they are in a presentable form. The simplest solution to this problem is to collect several events and package them together in a message before doing the multicast. However the space overhead of such a solution is going to be high for the following reasons.

Detailed Description Text (141):
The proposed solution, called Late Event Modification (LEM) enhances the event interface to allow applications to modify the event objects after posting them to the set. However, these modifications can only be made before the event is marshaled and sent out to other processes. For the purposes of explaining this application specific batching scheme, the following assumption is made about the client system implementation: There is a SharedEvent Thread 735 which maintains an event queue 730 for events waiting to be executed, and executes their handleEvent (FIG. 8) methods in order.

Detailed Description Text (143):
To support LEM, an abstract event class is provided called OpenEvent, which is extended (or sub-classed) by the application 103 to implement the required event functionality. FIG. 12 depicts an example of the OpenEvent class with the dashed box 1201 showing the abstract methods implemented by the application 103, and a monitor object 1210 which maintains a current state of an event. The abstract methods (the RealEvent interface 1201) are:

Detailed Description Text (145):
updateBeforeExec(Object arg) 1230: This handles any updates to the event Object before `handleSetEvent` 1202 is executed. Typically, this will just update the state of the event Object.

Detailed Description Text (146):
updateAfterExec(Object arg) 1204: This handles any updates after the handleSetEvent 1202 has been executed. This has to update the state of the event and the relevant shared objects.

Detailed Description Text (147):
As discussed, the monitor object 1210 maintains the current state of the event. An event generated at a local client 100 goes through three states after an application component 105 posts it to a shared set 680 and before it is marshaled and sent out to the server 120.

Detailed Description Text (151):
These are preferably concrete event classes. To use them, the application component 105 creates one of these event objects and issues it using the normal updateEvent method. It can then use the update method implemented by the OpenEvent class to modify the event object. This results in a call 1220 to updateBeforeExec 1203 if the event state is BEFORE 1211 or a call 1225 to update AfterExec 1204 if the state is AFTER 1212. If the state is DONE 1213, neither of these two event classes is called and a value of false is returned.

Detailed Description Text (152):
The OpenEvent class provides two mechanisms to control the transition to the DONE 1213 state (1) explicit, by invoking the done method 1231, and (2) implicit, by using a time-out 1232 value. The time-out value specifies the maximum number of milliseconds after the execution of the handleSetEvent 1202 that the event can stay in the AFTER 1212 state. Note, that this does not prevent the application from explicitly calling the done method 1231. Also note that the done method 1231 can be called explicitly by the application thread or can be used inside the RealEvent

1201 methods. For example, consider a text editor where every keystroke has to be echoed immediately. A keystroke will result in posting of a new event or updating the previous one. Suppose we want a transition to the DONE 1213 state whenever the user presses the `enter` key. This could easily be coded by making the updateBeforeExec 1203 and updateAfterExec 1204 methods check if arg=enter and then calling the done method 1231.

Detailed Description Text (153):
When this event is unmarshaled on a remote process, the monitor object 1210 is not created and the handleSetEvent 1202 is directly called 1241 from handleEvent. In fact, the abstract OpenEvent class has no state that needs to be marshaled.

Detailed Description Text (157):
The auctioning application has 2 types of application components 105: (1) an auctioneer; and (2) bidders. The auctioneer and the bidders share the following state using the shared objects 650.

Detailed Description Text (159):
The state of the item, i.e., is it sold or not sold and what is the highest bid on it and by whom.

Detailed Description Text (160):
The auctioneer is the only component 105 allowed to update this state.

Detailed Description Text (164):
This application demonstrates the natural way in which event sharing (FIG. 7) and state sharing (FIG. 10) can be combined. Note that the bidding and notification teams 230 could have been implemented with shared object 650, by having a shared bid object and notification object between the auctioneer and every bidder.

Detailed Description Text (167):
The application uses the conflict detection support (FIGS. 13, 14, 15, 16) provided by this invention to detect concurrent updates on the same object 650. It then allows the user to clone and export the complete set of objects to others (FIGS. 10-11). After deciding on the correct state, re-initialization of the complete set is done.

Detailed Description Text (174):
[2] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks, "Rendezvous: An architecture for Synchronous Multi-user Applications." CSCW 90: Proceedings of the Conference on Computer Supported Cooperative Work, ACM, pages 317-328 1990.

Other Reference Publication (10):
Y. Aahlad et al., "Asynchronous Notification Among Distributed Objects", USENIX Association, Conference on Object-Oriented Technologies and Systems, Jun. 17-21 1996, pp. 83-95.

Other Reference Publication (11):
K. Birman et al., "Lightweight Causal and Atomic Group", ACM Transactions on Computer Systems, vol. 9, No. 3, pp. 272-314, Aug. 1991.

CLAIMS:

1. A system for sharing application-specific events among a group of objects distributed over a network, comprising:

a plurality of clients comprising:

one or more distributable building blocks ("Components") of a collaborative application requiring support for shared state, said one or more Components

including one or more shared object sets, wherein each Component may be executed independently and collaborate with other Components; each Component comprising a set of one or more Event Reaction Units (ERU);

each ERU having an associated precondition and a reaction; wherein the precondition specifies one or more reference events that trigger the reaction, wherein the reaction may perform any computation or task and may produce other events;

cloning and reinitialization means for correcting diverging state among the shared object sets; and

a collaborative client middleware, communicatively coupled to the application and to the network, adapted to communicate to and receive from one or more remote clients over the network, asynchronous distributed shared events and deliver a received event to an associated Component.

2. The system of claim 1, further comprising:

context-sensitive state marshaling means for marshaling and unmarshaling clones of one or more shared objects in the shared object sets.

4. The method of claim 3 wherein the system is a client-server system, further comprising the steps of:

said receiving step comprising receiving from a client a post event to be posted to the team for the application; and

the team descriptor communicating the post event to the server, in response to receiving the post event.

6. The program storage device of claim 5, wherein the system is a client-server system, further comprising the steps of:

said receiving step comprising receiving from a client a post event to be posted to the team for the application; and

the team descriptor communicating the post event to the server, in response to receiving the post event.

7. A system for sharing application-specific events among a group of objects distributed over a network, comprising:

a plurality of clients comprising:

one or more distributable building blocks ("Components") of a collaborative application, wherein each Component may be executed independently and collaborate with other Components; each Component comprising a set of one or more Event Reaction Units ("ERU");

each ERU having an associated precondition and a reaction; wherein the precondition specifies one or more reference events that trigger the reaction, wherein the reaction may perform any computation or task and may produce other events;

a collaborative client middleware, communicatively coupled to the application and to the network, adapted to communicate to and receive from one or more remote clients over the network, asynchronous distributed shared events and deliver a received event to an associated Component; and

an event batching framework, comprising:

means for posting the event to the set of ERUs; and

a late event modifier ("LEM"), the LEM modifying the event objects after posting them to the set of ERUs.

8. A system for sharing application-specific events among a group of objects distributed over a network, comprising:

a plurality of clients comprising:

one or more distributable building blocks ("Components") of a collaborative application requiring support for shared state, said one or more Components including one or more shared object sets, wherein each Component may be executed independently and collaborate with other Components; each Component comprising a set of one or more Event Reaction Units ("ERU");

each ERU having an associated precondition and a reaction; wherein the precondition specifies one or more reference events that trigger the reaction, wherein the reaction may perform any computation or task and may produce other events;

a collaborative client middleware, communicatively coupled to the application and to the network, adapted to communicate to and receive from one or more remote clients over the network, asynchronous distributed shared events and deliver a received event to an associated Component; and

conflict detection means for detecting conflicts due to incorrectly ordered updates to a shared object and informing the application.

9. A system for sharing application-specific events among a group of objects distributed over a network, comprising:

a plurality of clients comprising:

one or more distributable building blocks ("Components") of a collaborative application requiring support for shared state, said one or more Components including one or more shared object sets, wherein each Component may be executed independently and collaborate with other Components; each Component comprising a set of one or more Event Reaction Units ("ERU");

each ERU having an associated precondition and a reaction; wherein the precondition specifies one or more reference events that trigger the reaction, wherein the reaction may perform any computation or task and may produce other events;

a collaborative client middleware, communicatively coupled to the application and to the network, adapted to communicate to and receive from one or more remote clients over the network, asynchronous distributed shared events and deliver a received event to an associated Component; and

context-sensitive state marshaling means for propagating state to one or more new or updated shared objects in the shared object sets.

11. In a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other Components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, a method for sharing events, comprising the steps of:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each

ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events;

detecting conflicts due to incorrectly ordered updates to a shared object; and

informing the application, in response to said detecting step.

12. In a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other Components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, a method for sharing events, comprising the steps of:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events; and

marshaling and propagating context-sensitive state to one or more new or updated shared objects in the shared object sets.

13. In a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other Components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, a method for sharing events, comprising the steps of:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events;

detecting diverging state among the shared object sets; and

cloning and reinitializing diverging state among the shared object sets.

15. The method of claim 13, wherein said cloning and reinitializing step further comprises the steps of:

making a copy of a subset of objects in the set and preserving pointers from inside the original subsets to objects outside the cloned subset;

mapping references between objects inside the original subset into the cloned subset;

exporting the cloned subset to other clients; and

re-initializing the subset of objects based on the cloned subset, in response to said exporting step.

17. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for sharing events in a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, said method steps comprising:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events;

detecting conflicts due to incorrectly ordered updates to a shared object; and

informing the application, in response to said detecting step.

18. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for sharing events in a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, said method steps comprising:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events; and

marshaling and propagating context-sensitive state to one or more new or updated shared objects in the shared object sets.

19. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for sharing events in a system wherein one or more distributable components of a collaborative application requiring support for shared state may be executed independently and collaborate with other components and wherein each component includes a set of one or more event reaction units ("ERU") and one or more shared object sets, said method steps comprising:

receiving a shared event from an application component over a network;

comparing the received event with one or more preconditions associated with each ERU, wherein the precondition specifies one or more reference events that trigger a corresponding reaction;

triggering the reaction, which may perform any computation or task and may produce other events;

detecting diverging state among the shared object sets; and

cloning and reinitializing diverging state among the shared object sets.

21. The program storage device of claim 19, wherein said cloning and reinitializing step further comprises the steps of:

making a copy of a subset of objects in the set and preserving pointers from inside the original subsets to objects outside the cloned subset;

mapping references between objects inside the original subset into the cloned subset;

exporting the cloned subset to other clients; and

re-initializing the subset of objects based on the cloned subset, in response to said exporting step.

First Hit Fwd Refs
**End of Result Set**

☐ Generate Collection Print

L18: Entry 2 of 2 File: USPT May 15, 2001

DOCUMENT-IDENTIFIER: US 6233565 B1
TITLE: Methods and apparatus for internet based financial transactions with evidence of payment

Abstract Text (1):
A system and methods for conducting Internet based financial transactions between a client and a server. The client has a processor, a printer, a client authentication module, a module for issuing a transaction request, and a unique digital signature. The server has a network including a transaction server, a transaction database, a server authentication module, and a receipt generation module. An internet connection is used between the client and the server network. The transaction execution system includes authentication, wherein the client authentication module and the server authentication modules communicate via the internet connection and are authenticated to each other. A transaction module is included wherein, in response to the client and server being authenticated, the client issues a transaction request to the server and the transaction server, in response to a client transaction request, executes an electronic payment transaction at the server and records the transaction in the transaction database. The server receipt generation module, in response to an executed electronic payment, then generates a receipt and transmits the receipt to the client. The receipt includes the client digital signature and a data set uniquely identifying the executed transaction and is printable by the client printer. The printed receipt is an evidence of payment for the executed transaction. In addition, a third party seller having a processor and a database can be connected via a communication channel to the server, wherein the client further obtains a registration certificate representative of being a consumer registered with said third party seller. A third party credit facility also may be connected via a communication link to the server, for implementing credit card transactions. The transaction execution system may be to purchase an amount of postage, to purchase a ticket for air travel or to an entertainment complex or the like.

Brief Summary Text (14):
Broadly, the invention concerns a system for electronic commerce including at least one user and a remote service provider, an Internet connection between each user and the remote service provider, wherein the user first becomes a registered user, e.g., registering with the remote service provider or with a third party supplier of goods and/or services or both, thereby obtaining a password set, and thereafter executes electronic transactions with the remote service provider using the password set to authenticate the user as a registered user and the remote service provider as an authentic service provider, and receives a secure evidence of payment for each transaction executed including a digital signature and data uniquely identifying the transaction.

Brief Summary Text (16):
The customer and potential user then uses the provided account information to download the appropriate portion of the system software (also referred to as the "client" software) from the RSP. The potential user then installs the client software on a local PC type device and configures the software and hardware of the

system for use including establishing a transaction database specific to the user. This database may include a Register indicating purchases or funds available or the like. The system also provides for authenticating the user to the RSP and the RSP to the user before any transaction can occur.

Brief Summary Text (17):
After authentication is completed, the user then purchases the ultimate goods or services, postage in the case of the preferred embodiment, utilizing credit cards, ACH debit cards or checks as the method of payment, and electronically confirming the sale. The RSP then operates on the user's transaction database, e.g., increments a "descending" register, associated with the specific user corresponding to postage purchased by the user. The transactional database for the user is maintained by the RSP and uses special transactional software, which for postage is referred to as a Postal Security Device (PSD) software, resides on a server of the RSP, and manages the accounting, auditing, and security, digitally signing each transaction to identify uniquely both the user and the transaction. The special transaction software also manages the printing at the user of evidence of payment of the purchase, e.g., postage, postage corrections, and refunds, and other miscellaneous communications with the RSP as appropriate and the TPS of goods or services, in the case of the preferred embodiment the United States Postal Service. Should there be a problem with the authentication process, the RSP server would disable the descending register and report the failure to the appropriate USPS authority, e.g., the National Meter Accounting and Tracking System ("NMATS"). Should there be a problem with the user's registration/license or should fraud be detected, e.g., detecting the same evidence of payment more than one time, the RSP server would disable the descending register and report the failure to NMATS. Should the user wish to terminate use of the product, the client software locally downloaded can be automatically uninstalled, disabling and rendering inoperable all aspects of the system software for the user. Uninstalling the client software would result in revocation of the customer's license. This revocation will be reported to the appropriate authority, e.g., the Centralized Metering Licensing System ("CMLS"). Any subsequent use of the "retired" password or account numbers by that or any other user constitutes fraud, which is detectable.

Brief Summary Text (19):
Another aspect of the present invention is directed to a postal purchase system. One such system includes, for each customer, a client system and a Postal Secure Device (PSD) as defined by the IBIP specifications. The client system is a Host which resides on the customer's local PC and is responsible for the following: mailing list management, capturing postal purchase and refund request information, and providing an interface to the local printer producing the postal indicia. Each customer's PSD resides at a RSP "server" site and can be accessed only via the Internet. The customer's PSD is responsible for managing ascending and descending registers which track postal transactions, and creating a digital signature for each postal indicium produced by the customer on the local printer. By keeping all PSD functionality on a remote, central station server/network, all cash and key management and process auditing can be centralized and secured.

Brief Summary Text (21):
a client having a processor, a printer, a client authentication module, a module for issuing a transaction request, and a unique digital signature;

Brief Summary Text (22):
a server having a network including a transaction server, a transaction database, a server authentication module, and a receipt generation module; and

Brief Summary Text (25):
an authentication module, wherein the client authentication module and the server authentication modules communicate via the internet connection and are authenticated to each other;

Brief Summary Text (26):
a transaction module wherein, in response to the client and server being authenticated, the client issues a transaction request to the server and the transaction server, in response to a client transaction request, executes an electronic payment transaction at the server and records the transaction in the transaction database, and wherein the server receipt generation module, in response to an executed electronic payment, generates a receipt and transmits said receipt to the client, said receipt comprising the client digital signature and a data set uniquely identifying the executed transaction; and

Brief Summary Text (29):
The server may include a first server, a firewall and a single TCP/IP port, such that the first server is connected to the internet connection, the firewall is interposed between the first server and the transaction server and the transaction server is connected to the network through the single TCP/IP port. In this embodiment, the firewall comprises a communication module that operates to limit communications between the internet and the transaction server to client transaction requests identifying the single TCP/IP port.

Brief Summary Text (33):
Regarding authentication, the client may have a client public key, a client private key, and a client identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the client public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the client private key and client public key based on said hash. Similarly, the server may have a server public key, a server private key, and a server identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the server public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the server private key and server public key based on said hash. Thus, the authentication module utilizes the public and private keys of said client and server to perform the authentication.

Brief Summary Text (35):
(a) providing a client having a processor, a printer, a client authentication module, a module for issuing a transaction request, and a unique digital signature;

Brief Summary Text (36):
(b) providing a server having a network including a transaction server, a transaction database, a server authentication module, and a receipt generation module;

Brief Summary Text (38):
(d) authenticating the client and the server to each other;

Brief Summary Text (40):
(f) in response to a client transaction request, executing an electronic payment transaction at the transaction server and recording the transaction in the transaction database, generating a receipt at the server receipt generation module, providing said receipt with the client digital signature and a data set uniquely identifying the executed transaction, and transmitting said receipt to the client; and

Brief Summary Text (43):
In a preferred embodiment, step (b) includes providing a single TCP/IP port connecting the first server to the internet connection with the client and limiting communications between the internet and the transaction server to client

transaction requests identifying the single TCP/IP port.

Brief Summary Text (45):
In a preferred embodiment, step (d) includes providing the client with a client public key, a client private key, a client identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the client public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the client private key and client public key based on said hash; providing the server a server public key, a server private key, a server identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the server public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the server private key and server public key based on said hash; and using said public and private keys of said client and server to perform the authentication.

Drawing Description Text (9):
FIGS. 6A and 6B are a block diagram of an authentication process for use in the present invention;

Drawing Description Text (10):
FIG. 7 is a block diagram of the architecture an authentication process of the system of FIG. 1; and

Detailed Description Text (5):
The infrastructure of the RSP (or "server") 4 is preferably designed to address the following goals: (1) It will meet all Level 2 security requirements as defined in the aforementioned FIPS 140-1 and FIPS 186 specification, except for physical security because the transaction server(s) 180 on which the PSDs 20n and Master PSD 40 reside will exist on a protected segment behind a firewall 160 (see FIG. 2). Security measures are used to ensure that the transaction server 180 is physically accessed by highly trusted and authorized individuals only; (2) It will be scalable from zero to two million customers (n, n=1 to 2.times.10.sup.6 customers); (3) Customer Host 10n requirements are a personal computer ("PC") with a 486 processor or higher, 8 Megabytes of RAM, a hard drive with at least 10 Megabytes of space available, a modem (9600 BPS or higher), a printer (laser, inkjet, or bubblejet), running preferably Windows95.TM., a web browser software, and an active Internet connection; (4) Non-print electronic transactions should be performed in less than five seconds; and (5) Print electronic transactions should be completed within thirty seconds.

Detailed Description Text (6):
Referring to FIG. 2, the infrastructure of RSP 4 can be broken down into several pieces: (1) A web server 150 that is used by customers for registration and client software download and is the apparent website; (2) A transaction server(s) 150 that serves as the transactional link between the customer's Host PC 10n and all RSP 4 functions; and (3) A series of database servers 170 that perform all of the RSP 4-related and TPS 6-required functions.

Detailed Description Text (7):
Referring also to FIG. 2, to protect the various components from unauthorized access and intrusion, the PSP 4 is provided with an inbound network 110 and an outbound network 130. The inbound network 110 allows a customer 2n to securely access the RSP web server 150. The outbound network 130 allows for the secure exchange of financial transactions executed between the customer 2n and the TPS 6 directly and/or indirectly through RSP 4. Network traffic in and out of these networks is controlled by a conventional inbound router 112 and an outbound router 132, which will operate to filter out all unauthorized traffic. In addition, a firewall 160 will be used on the inbound and outbound segments to examine each data packet transmitted for proper authorization. The secured portion of the RSP web

server 150, i.e, that portion which is access protected by passwords to authorized/registered users, will exist on a unique port 140 so that only traffic identifying that specific port 140 will be accepted. The transaction server(s) 180 will exist on a unique internet protocol ("IP") address(es) so that the outbound router 132 can filter out all traffic except to that address. The transaction server 180 will also be configured to handle IP traffic only. The outbound router 132 and the firewall 160 will filter out all other Internet protocols according to industry standards for such firewalls. The PSP 4 also includes a series of data servers 170 (shown collectively in FIG. 2) that will be responsible for various dedicated functions. As the transaction server 180 receives a transaction request, the type of transaction is identified and processed using one of these servers as identified in the Table I below.

Detailed Description Text (21):
During the Registration process, the web server 150 obtains and forwards the appropriate information to TPS 6, e.g., the United States Postal Service Certifying Authority for issuance of the user's certificate ("X.509") as required by Postal Service specifications. Preferably, the issuance of the Postal User Certificate X.509 from the United States Postal Service Certifying Authority will be accomplished in as near to real time as possible, because generally a postage value download to the user's PSD 20n will not be permitted without an authenticated certificate.

Detailed Description Text (22):
With reference again to FIG. 3 and to FIG. 6, after receiving the certificate and a license/registration number at step 216, the user 2n can then proceed to make a purchase, e.g., of postage. User 2n makes a purchase through a proprietary connection over the Internet 30 using the appropriate IP address as provided by the downloaded client software to connect with the RSP's Internet transaction server 180, utilizing a suitable form of payment, such as credit cards, electronic funds transfer, ACH debit cards, or checks.

Detailed Description Text (23):
Electronic payments are reported to the transaction server 180 which then transmits them to a specified financial institution for deposit. Upon receipt back of an authorization code, transaction server 180 then increments the user's descending register 21 in PSC 20n with the correct amount. The ascending and descending registers of each user 2n are stored on the Master Server 300 at RSP 4. Under current Postal Service guidelines, the total maximum amount permitted in the descending register 21 is $500.00, but any value could be used, as well as no limit at all.

Detailed Description Text (24):
Checks used to pay for postage are preferably sent by the user 2n to a designated lockbox institution for processing in a conventional manner. When a user's check has cleared, the lockbox institution transmits an electronic authorization to the transaction server 180, which operates to transmit a notice to the user 2n of postage availability. The user 2n typically must then access the transaction server 180 to obtain the postage (which may have already been allocated to the user 2n and held in a suspense server 310) and server 180 operates the master server 300 to increment the user's descending register 21n in PSD 20n by the proper amount and clear the register in suspense server 310.

Detailed Description Text (31):
A portion of the system software resident on the transaction server 180 of RPS 4 functions as the Postal Secure Device (PSD) 20n. This software portion, known as the Transaction Manager, encompasses many features and benefits to both the TPS 6 (e.g., United States Postal Service) and customers, as illustrated by Table II.

Detailed Description Text (36):

The system's security features will use standard, publicly available NIST-approved algorithms. These are discussed below under security, authentication and authorization.

Detailed Description Text (40):
All purchase and refund requests will be digitally signed and encrypted for transmission from the host IOn to the transaction server 180. RC2 symmetric encryption standard key pairs (public key/private key) may be used to support such encryption and decryption. RC2 will be used to protect the nature of the purchase/refund request, which may include credit card information. RC2 is a well-known industry standard. RC2 is a product of RSA. RSA is an accepted vendor for these products according to the IBIP Indicium specification dated Jun. 13, 1996.

Detailed Description Text (41):
For credit card purchases/refunds the transaction from the transaction server 180 to the credit bureau 9 preferably will employ SSL3.0 (Secure Socket Layer) standard encryption for secure packaging of the transaction. SSL3.0 is an accepted industry standard for financial transactions across a TCP/IP network. Other suitable encryption standards could be used.

Detailed Description Text (60):
A function allows the user 2n to purchase postage from the PSP 4 transaction server 180. It is initiated by selecting a "Purchase Postage" screen in the client software. The screen displays the maximum postage that can be purchased, any fees associated with this purchase, which are charged by PSP 4 or TPS 6 as the case may be, and the total cost for the purchase. The customer 2n must enter an amount of postage to purchase and select a method of payment. The methods of payment are preferably shown with "radio" buttons and include credit card, ACH debit, and check. The default payment is preferably by credit card and the credit card input area is activated.

Detailed Description Text (61):
The customer then initiates transmission of all of the purchase information (e.g., addresses, purchase amount, and credit/check information) via the Internet 30 to the web server 150, which passes on the transaction request to the transaction server 180. When the Submit radio button is pressed, all customer information is digitally signed and encrypted and packaged with the purchase amount. A connection over Internet 30 is established with the web server 150 and the transaction related information is then transmitted. Because the transmission has the appropriate IP address for the transaction server 180 it will be directed by web server 150 through the firewall 160 to transaction server 180, where the transaction will be executed. If a connection cannot be made with the web server 150, then control is passed to an Unable To Connect Error. When the web server 150 responds, the text message received from the server 180 is displayed in the main message area at the top of the screen of the user's Host 10n. Possible responses include Purchase Complete, Incorrect Credit Card Information, Purchase Pending, or Credit Denied. If a Purchase Complete or Pending message is received, the descending register 21 (Postage Remaining field) specifically associated with the customer 2n in the PSD 20n is updated with the new value from the transaction server 180. The actual PSD registers 20n remain on the master server 300; the Host 10 merely displays a copy of the PSD 20n register 21 values.

Detailed Description Text (65):
After the user fills out the entire field on the purchase screen and selects a submit radio button, the transaction server 180 immediately constructs a new purchase request object base from these field values.

Detailed Description Text (66):
After the transaction server 180 receives the purchase request, it interacts with the following servers to execute the transaction:

Detailed Description Text (68):
2. Purchase server 190 (after the purchase request object is deciphered by the transaction server 180, it passes to the purchase server)

Detailed Description Text (69):
Credit card requests are transmitted to the web server 150 by the client, forwarded to the transaction server 180, and then to a payment server 190, a credit authorization server 400, and to a remote credit bureau 9 such as to First Data Merchant Services ("FDMS"). The credit authorization server 400 is responsible for connecting to the credit card bureau 9 and getting approval: A "result" code is passed back to the purchase server 190 to indicate whether the credit card has been approved or not. For example, 0 means on the credit link is down (after 3 attempts), 1 means the credit card was rejected, and 2 means the purchase is approved.

Detailed Description Text (71):
If payment is by check, the check number is sent to the web server 150 and purchase server 190 along with the information listed above. The response from the purchase server 190 will include a customer number. When the response is received at the Host 10n from the transaction server 180 then control is passed to the Remittance Pop-Up Window.

Detailed Description Text (77):
The customer then must enter in credit card, debit card, or check information so that the appropriate account can be credited or a check can be issued. The user presses the Submit button to submit the request to the web server 150 and then to the transaction server 180. For a spoilage refund, a copy of the updated descending register 22 is received from the server 180. The actual descending register 22 is updated on the server 20n.

Detailed Description Text (80):
In each of the foregoing transactions, once a transaction is completed, transaction server 180 creates a response transaction which is digitally signed with the digital signature of the customer 2n requesting the transaction, encrypted, and sent to the Host 10n confirming the success or failure of the transaction. The host then updates its local information to reflect changes in postage available.

Detailed Description Text (91):
With respect to maintaining logs databases 196, 197 of indicia files on the transaction server 180, it is expected that the United States Postal Service will permit these files to periodically be off loaded onto tape, or otherwise transmitted periodically. See, e.g., FIG. 1A, the reference to "batch" connecting data management functions 42 and 62.

Detailed Description Text (94):
Preferably, the system permits the user to preview a single envelope or label by pressing a Print Preview button. This will cause the return address, the mailing address, and a bitmap of a sample indicium 74 to be displayed as it would appear printed. Pressing this button will pass control to a standard Print Preview screen. If the user wishes to print the entire list, the Print All button is selected. Pressing Print All causes a connection to be made to the web server 150 and a file of addresses sent. This file is digitally signed by the client Host 10n for the transaction server 180 to verify. On the transaction server 180, the digital signature on the file will be verified by validation (also called the security) server 315, each address will be extracted individually and the contents of that address, along with additional information, will be used to create both the human- and computer-readable parts of the indicium 74. See indicium generation 43, FIG. 1A. The postage rate table used is checked to see if it is current, and the postage amounts are recalculated if needed. The contents of the indicium are then hashed

(MD5) into a message format and the resulting message is digitally signed by transaction server 180 using the well-known DSA. The indicium is then encrypted for the client to decrypt. The indicium is encrypted to prevent unauthorized capture of the indicium. An unencrypted indicium could be captured, printed, and entered into the mail stream prior to it being obtained by the authorized customer.

Detailed Description Text (103):
The present invention advantageously uses the internet 30 connection for registration, client/server authentication, transmission of credit card information, transport of indicia, requests for refunds, change of personal/address information, and the exchange of addresses for cleansing. Each of these transactions will require different types of security to ensure the safe exchange of information between the Host 10n, PSD 20n, and printer 70n. Conducting financial transactions over an unsecured channel such as the internet 30 requires the use of cryptographic modules. In the present invention, each client 2n has a cryptographic module 12n and the RSP server 4 has a cryptographic module 14. The server cryptographic module 14 serves three functions: (1) authentication, (2) encryption, and (3) authorization. Authentication is the only function that requires interaction with a client cryptographic module 12. This is discussed below.

Detailed Description Text (104):
The physical architecture of the system of the present invention was designed to ensure that all access to the system is through secured and monitored points. The entire system that is at PSP 4 will reside on a private network and the transactional portions will be connected through a firewall 160 to the internet 30. Firewall 160 will be configured to restrict all network traffic to a single TCP/IP port 140. All packets received by the firewall 160 on the specified port will be routed to a Transaction Manager (i.e., transaction server 180. It is noted that the actual implementation may have a plurality of transaction servers 180n, and routing a given packet to a transaction server 180 may be based on an address field in the packet or on a first available server, as the case may be. Once a transaction server 180n receives the packets that request a socket connection, a socket connection will be established. The transaction server 180 will immediately require that each particular connection authenticates a client 2n, or the connection will be dropped. To ensure this safe exchange, the implementation of security uses the following assumptions:

Detailed Description Text (105):
The host 10n and PSD 20n functions will not reside on the same machine. The PSD 20n functions will not be stored in a separate hardware device connected to the customer's PC. The Host 10n will exist on the customer's PC and the PSD 20n will exist on the PSP 4 network infrastructure. The Internet 30 will be used to interconnect a customer's host 10n and printer 70n with the PSD 20n. All transactions between the Host 10n and the PSD 20n will be encrypted. All transactions between the host 10n and the PSD 20n will be digitally signed. All indicia 74 will be digitally signed. All indicia 74 will be encrypted for client 10n to decrypt prior to printing. Prior to initiating a transaction, both the client 10n and the provider transaction server 180 will authenticate each other. A customer's actual existence and proof of valid physical address will be initially established by sending the system license and registration information to the customer by mail. All cash management functions will be performed within the treasury component of the RSP 4, namely, payment server 190, separate from the PSD 20n functionality.

Detailed Description Text (106):
The following discussion describes a preferred security model to be implemented. The downloaded client software performs all Host 10n and printer 70n functions, as defined by the appropriate IBIP specifications. The transaction server 180 performs all of the PSD 20n and TPS 6 infrastructure and support functions, as outlined by the IBIP specifications. Each client 2 and server 4 is comprised of its own

cryptographic module 12 and 14, respectively. For authentication and key expiration/regeneration the two modules 12 and 14interact.

Detailed Description Text (108):
The cryptographic module 12 is used to authenticate the customer 2n to the TPS 4 (hereinafter also referred to as "server 4"), the server 4 to the client 2n, and to manage the authentication key pair (public key/private key) that exists on the client 2. The main function of the client cryptographic module 12 is to protect the customer's private key from both intrusion and corruption. The customer's private key is used to authenticate the client 2 to the server 4.

Detailed Description Text (116):
The most important responsibility that the CryptoManager assumes revolves around protecting a file known as the Key File. The Key File will contain the necessary information to uniquely identify and authenticate a client 2n to the server 4. It is composed of four items. 1) The Client's Public RSA Key; 2) the Client's Private RSA Key; 3).the Customer Identification Number; and 4) A Digital Hash of 1, 2, and 3.

Detailed Description Text (123):
The key file is preferably provided a mandatory life span that will be enforced by the Host 10n's cryptographic module 12. After a designated period of time, the Host 10n will open an interface 530 that will require the client to re-generate a new pair of public and private keys 532. The client will generate a new key, using a standard Microsoft CryptoAPI.TM. function, calculate a hash of the key file, encrypt the key file using the first 64 bits and store the generated key pair as ciphertext keys 535. The client 2 will then send the new public key to the web server 150. The client cryptographic module 2 will authenticate with the cryptographic module 14 of server 4 to verify that the key-renewal process was successful.

Detailed Description Text (126):
vi. Authentication

Detailed Description Text (127):
The authentication service is responsible for authenticating each client 2n with the server 4. As illustrated in FIG. 7, a sequence of three messages is transmitted between the client 2n and the server 4. E.sub.spu refers to RSA encryption using the server's public key. E.sub.cpu refers to RSA encryption using the client's public key. In the case where the client 2n fails to authenticate with the server 4, the authentication protocol is terminated and the communication link between the client 2n and server 4 is severed. If the client 2n is able to successfully authenticate with the server 4, then the session keys exchanged and established during authentication can then be used to encrypt communications between the client 2n and server 4. Authentication is discussed more fully below in connection with server 4.

Detailed Description Text (129):
After authentication, the client 2n executes a HouseKeeping Service. This Housekeeping service is responsible for notifying the client software of any new postal table information or changes and of key expiration events. The HouseKeeping Service queries the server 4 to ensure that current keys have not expired. If the keys are about to expire the client 2n will issue a command to renew the keys and a new public key will be sent back to the server 4. The client 2 must then re-authenticate with server using the new keys. If the keys have not expired, then the HouseKeeping Service will check the postage rate tables. If the postage tables need to be updated, the client 2 will submit a request to server 4 for new postage tables. If the keys have not expired and postage tables do not need to be updated the HouseKeeping service exits.

Detailed Description Text (136):
The Crypto Officer State 940 is entered only when the key file needs to be updated. This will happen either when the user changes the password or when the current RSA keys expire and need to be changed. The server 4 during authentication will trigger replacement of expired keys.

Detailed Description Text (140):
In addition to protecting sensitive data, the server cryptographic module 14 runs many security services. These services include client authentication, key management and PSD management. The security services are governed by policies that dictate who can access them and what data they can control.

Detailed Description Text (141):
The client 2 has no direct control over the services performed by the server cryptographic module 14. All services performed by the server cryptographic module 14 are under the direct control of a Transaction "Manager" server 180. Once the client 2 has been authenticated, it submits a transaction request to the transaction server 180 and waits for a response. It now becomes the job of the Transaction Manager to process the transaction and return a "receipt" to the client 2. All transaction "receipts" will contain a date/time stamp, and a sequence number and a digital signature to verify the authenticity of a transaction in relation to other similar transactions. For a given customer 2n, the sequence number will increase by one each time until a threshold is met, e.g., 10,000, at which point the counter will reset to 1.

Detailed Description Text (142):
The Transaction Manager embodies the logic required to complete a transaction while enforcing the security and integrity of the server cryptographic module 14. Based on the authenticated user 2n, the transaction requested, the date/time stamp, and the sequence number, the Transaction Manger would determine whether the requested transaction is valid. The Transaction Manager will complete the transaction by sending a receipt, including a digital signature as evidence of payment for the transaction, back to the client 2n. In the case of the postage purchase system, the receipt is the indicium 74, which includes the foregoing and the postage related information (addresse, postage amount, etc.).

Detailed Description Text (143):
The Transaction Manager server 180 also is responsible for performing non-cryptographic functions. For example, address cleansing, credit approval and customer profile changes, which are performed by transaction server 180, do not require the use of cryptographic functions, and are therefor considered outside the realm of the server cryptographic module 14.

Detailed Description Text (144):
The server cryptographic module 14 serves three essential functions authentication, encryption and authorization. Authentication is the only function that requires interaction with the client cryptographic module 12. This interaction is secured in a manner described elsewhere.

Detailed Description Text (150):
The Cryptographic keys are utilized by the cryptographic module 14 to perform two main functions, authentication and indicium generation. The cryptographic module stores the following keys. Client Public Authentication Keys (RSA) A Client Public Authentication Key exists for every registered user. They are used to prove the client's identity when it attempts to establish a connection with the server 4. This will be typically a 1024 bit key. Server Public/Private Authentication Key Pair (RSA) Only one pair of Server Public/Private Authentication Keys exists on the server 4. These are used to prove the server's identity when the client 2n attempts to establish a connection with the server. This will be preferably a 1024-bit key. Client Private Indicium Keys (DSA) A Client Private Indicium Key exists for every

registered user 2n. It is used to generate the digital signature required to produce an indicium 74. This will be a 1024-bit key. Server Internal Private Key (DSA) This key is used to sign data that will be used to communicate with the TPS 6, in this embodiment, the USPS. This will be a 1024-bit key.

Detailed Description Text (153):
The certificates used by the cryptographic module 14 essentially complement the private keys described in the above section with the exception of the Server Authentication Keys. The Client Indicium Certificate is the data used to generate the digital signature of the indicium. The actual signature is signed with the complementary key of the certificate's public key. This way, based on the contents of the certificate, an exterior entity can verify authenticity of the generated indicium 74. The Server Certificate is used by the USPS to confirm the server's identity. The USPS requires that the server 4 periodically communicate with it to transfer information regarding postage sold. This certificate is used to ensure safe delivery of this information. The Certification Authority Certificate issued by the Certificate Authority allows the server 4 to verify messages sent by various other parties that have been certified by the same Certification Authority.

Detailed Description Text (174):
Authentication Service

Detailed Description Text (175):
The authentication service is responsible for ensuring that only authorized users 2n have the ability to submit transaction requests. It will utilize the client's public authentication keys to perform the user validation. Upon successful authentication the transaction manager (server 180) will handle the user's transaction request.

Detailed Description Text (183):
The cryptographic module 14 supports the use of two roles to perform different cryptographic functions. The first role is the Crypto-Officer that is responsible for performing all the key management functions. The other role is the Crypto-User, which has access to use cryptographic functions and keys for the purposes of authentication and indicium generation. The roles provide a logical separation between operators and the services they are allowed to perform.

Detailed Description Text (185):
Authentication and Encryption

Detailed Description Text (186):
When a client 2n establishes a connection with the server 4, the server 4 immediately enters an authentication protocol as illustrated in FIG. 7. The first step of the routine is to authenticate the client 2n. The client 2n sends its customer ID in plaintext form. This information is used to verify that this is a valid customer and retrieve that customer's public key. The client 2n creates a random session key, identified in FIG. 7 as Session Key A. The client 2n then uses the server's authentication RSA public key to encrypt Session Key A, and transmits the encryption to the server 4. The server 4 receives and decrypts the key using its private authentication key.

Detailed Description Text (187):
The server 4 then creates a random session key of its own, identified in FIG. 7 as Session Key B. The server 4 uses the client's public authentication key to encrypt Session Key B, and transmits the encryption to the client 2n. The client 2n receives and decrypts the key using its private authentication key.

Detailed Description Text (190):
Once the client 2n and server 4 have exchanged session keys and hash values and the hash values have been properly validated, the protocol is complete and the client

2n and server 4 have been authenticated to each other. Session Key A and Session Key B can then be used by the client 2n and server 4, respectively, to encrypt and secure communications to each other.

Detailed Description Text (191):
The session keys generated during the authentication process are preferably 64-bit RC2 symmetric keys, which are used to encrypt and decrypt all data sent between the client and the server. A hash of the data value is appended to all transaction request data or a transaction receipt before it is encrypted and sent to the other party. The recipient uses the hash to ensure message integrity. If the message is found to be corrupt or altered in some way, the transaction will be aborted.

Detailed Description Text (192):
At the end of the Authentication protocol, the client cryptographic module 12 will store the Customer ID and the session keys as part of the initialization routine. This information will then be available to all services running under Crypto-User role for a particular user.

Detailed Description Text (194):
The transaction server 180 is concerned with two types of public/private key pairs, authentication and indicium generation. Authentication keys are 512-bit RSA keys and the indium generating keys will 1024-bit DSA keys. Both the client 2 and the server 4 each have a pair of authentication and indicium generating keys.

Detailed Description Text (195):
Client Authentication Keys

Detailed Description Text (196):
The server 4 initially produces the client authentication keys. The server first generates the client's public/private RSA 1024-bit keys. It then bundles the client keys, the server's public key and the client's identification number into a single file. A MD5 hash of the file is produced and appended to the end of the file. Finally, the server 4 takes the content of this file and encrypts it with a random number that it generates. The produced ciphertext is referred to as the key file. The server generated password initially used to encrypt the key file is mailed to the client, as explained in the registration procedure described above. The key file is then exported from the server's cryptographic module to a SSL 3.0 password secured web site 150. The customer can then change this password in a subsequent session.

Detailed Description Text (197):
The client authentication keys will have an expiration period. The duration of this period will be, for example three (3) years, as determined by the USPS Key Management Specification. To prevent checking for key renewal every time a client 2n connects to the server 4, all client keys may be set to expire at the same time. After a client 2n authenticates with the server 4, the server 4 will notify the client if its keys have expired. At that time, the client will generate a new set of keys and send the public key to the server. The server only stores the client's previous public authentication key. The private key is kept private with only the client. The public key is kept inside a password protected SQL master database 305 (FIGS. 4, 7) that is accessible only by the server cryptographic module 14. In the scenario when a user 2n requests a full refund, that user's keys are destroyed and their record removed from the SQL database 305.

Detailed Description Text (198):
Server Authentication Keys

Detailed Description Text (199):
The software product will execute a key generation routine. This routine will produce the server's public/private RSA 1024-bit authentication keys. The key will

be stored on a secured SQL master database 305. The public/private keys will have an expiration period defined by the USPS. When the server's keys expire, it will regenerate a new pair of authentication keys. With every client record, a field will be used to indicate which version of the server public key its using. During authentication, the server 4 will retrieve the appropriate version of its authentication public key. If the client 2n is using an expired key, the server 4 will download the new key to the client 2n. The client's record will be modified to reflect the new key version. The server 4 will keep its old authentication keys archived for a specified period of time. This allows a client 2n who has not connected since the server's public key expired the ability to authenticate and download the server's current public key.

Detailed Description Text (201):
The Certification Authority (CA)'s certificate is the only self-signed certificate on the system. The public key in the certificate corresponds to the private key used to sign the certificate. This makes the Certification Authority certificate easily susceptible to fraud if proper distribution mechanisms are not employed. The initial CA's certificate will be distributed by means of regular US certified mail. Included with the CA's certificate will be a hash of the next certificate key values. When a certificate expires, the USPS certification authority will issue a new certificate and sign it with the old certificates matching private key. The USPS CA will send a new certificate signed with the CA's new private key to the server 4. The server 4 will validate the certificate for authenticity by first checking to ensure that the new CA certificates public key authenticates the included signature. It will then hash the keys included with the new certificate to verify that the hash value match with the old hash included with the old CA's certificate. If both conditions validate, the old CA's certificate is deleted and replaced with a new CA certificate.

Detailed Description Text (205):
The server 4 needs a public/private key pair for each client 2n in order to produce indicia. The server 4 will generate a key pair for the client 2n, store the private key in the secured SQL master database 305 and send the public key to the CA as a request. The request is signed with the server's private indicium key and sent to the CA, where the request will be authenticated using the server's certificate. The CA will send the certificate back to the server. The returned certificate will be packaged with some other information and digitally signed by the CA. Using the CA certificate the server 4 will verify the authenticity of this returned package.

Detailed Description Text (216):
Encrypt (PlainText [in], CipherText [out]) This interface will encrypt all the plaintext that it receives as an argument and outputs it in ciphertext. The cryptographic algorithm will be 64-bit RC2. The key used in this function remains inside of the cryptographic module, it is generated by the authentication routine.

Detailed Description Text (217):
Decrypt (SessionKey [in], CipherText [in], PlainText [out]) This interface will decrypt all the ciphertext that it receives as an argument and outputs it in plaintext. The cryptographic algorithm will be 64-bit RC2. The key used in this function remains inside of the cryptographic module, it is generated by the authentication routine.

Detailed Description Text (219):
Authenticate (OpenSocket [in], status [out]) The authentication interface will be called to verify and authenticate the identity of the client 2n. It is passed a handle to the current open socket of the client requesting a connection to the server. The status will indicate whether the client has successfully authenticated.

Detailed Description Text (223):

GenerateServerAuthenticationKevs (status [out]) This will be an administrative function that will generate a new authentication key pair for the server 4. The status will indicate whether the function completed successfully.

Detailed Description Text (224):
StoreClientPublicKey (PublicKey [in], status [in]) This interface will allow the client's public authentication key to be updated in the cryptographic module's database. The status will indicate whether the function completed successfully.

Detailed Description Text (227):
Referring to FIG. 8 again, the finite state model defines the set of system access rules for the server cryptographic module 14. The model defines secured and unsecured states and all state transitions. When the software is executed, the cryptographic module 14 enters the Self-Test State 910. For example, the first test will verify that the executable module check-sums are correct. This will ensure that none of the code has been corrupted or modified. Next, the cryptographic algorithms will be tested. The RSA, RC2 and MD5 cryptographic algorithms will be tested using the "known-answer" test. A known value will be applied to the algorithm to determine if it will reproduce a known result. If the resulting value matches the expected result, the algorithm is assumed to be functioning properly. Since the DSA algorithm does not reproduce the same value twice, it will use pair-wise consistency test. This test will first sign a quantity with a private key and then verify the signature of this quantity using the public key. If the result of the verification is successful, the test succeeds. If any of cryptographic modules tests fail, it is assumed to be malfunctioning. If all self-tests pass successfully, the module proceeds to enter the Un-Initialized State 920, otherwise the module will enter into the Error State 950. No keys are loaded into the module during the Un-Initialized State and therefore no cryptographic functions can be performed. It is in this state that authentication service starts. Based on the authenticated user and the requested transaction, the cryptographic module will determine whether the next state should be the Key Entry State 960 or the Crypto-officer State 940. This decision will depend on whether the transaction will use the cryptographic module to perform key management or other cryptographic functions.

Detailed Description Text (231):
In the Key Entry State 960 the cryptographic module 14 loads the keys obtained from SQL master database 305 for the authenticated user 2. The Initialize function of the Crypto-API library is used by the cryptographic module as a means of initializing the crypto-context with the keys. After successfully loading the keys, the module 14 moves to the Idle State 970.

Detailed Description Text (238):
Set forth below in Table IV is a table of the various security features used in the functions performed by Transaction Server 180.

Detailed Description Text (257):
The Audit Function required by the United States Postal Service can be satisfied by various functions of the system working synergistically under the control of Housekeeping Services. The Housekeeping service is responsible for up-loading configuration changes and new logs to the server 4. It also facilitates downloading of new client software and pricing changes from the server 4 to the client 2. This service is preferably called every time the user 2n connects to the server 4 and is called upon immediately after the Authentication procedure.

Detailed Description Paragraph Table (1):
TABLE I Transactions Server Actions Transaction Master Server Payment Server Postage Server Customer Server Credit Auth. Transaction Type Server 190 300 190 Log Server 195 450 460 Server 400 Authentication 1. Verify client 1. Provide 1. Log Transactions Provide customer client 2n to server Signature customer information 2.

Decrypt public key client request 2. Send 3. Request Transaction customer Summary to lookup Log Server 4. Decipher message 5. Validate customer 6. Send digitally signed and ciphered message Purchase of 1. Verify 1. Increment Manage 1. Log Transactions 1. Decrement 1. Issue Good/services e.g., digital; customer PSD Purchase 2. Send Log to Master check request postage Signature of ascending Process TPS (IBIP descending 2. Receive client register Infrastructure) register response 2. Decrypt 2. Send 2. Request 3. Send response client Transaction postage from to Purchase Request Summary to TPS (USPS) Server 3. Send Log Server 3. Increment purchase Master Request to descending Payment register Server 4. Send 4. Digitally Transaction Sign and Summary to Encrypt Log Server Response 5. Send ciphered message to client Refund 1. Verify client 1. Increment Manage 1. Log Transactions 1. Decrement 1. Issue credit Signature customer PSD Refund 2. Send Log to TPS Provider PSD check request 2. Decrypt ascending Process (IBIP ascending 2. Receive client register Infrastructure) register response Request 2. Send 2. Send 3. Send response 3. Send Transaction Transaction to Purchase Request Summary to Summary to Server to Payment Log Server Log Server Server 4. Digitally sign and Encrypt Response 5. Send digitally signed and ciphered message to client Print 1. Verify client 1. Decrement 1. Create 1. Log Transactions Signature customer PSD Indicia 2. Send Log to 2. Decrypt ascending 2. Create file: IBIP client register of Indicia Infrastructure Request 2. Send 3. Notify 3. Send Transaction Transaction Request to Summary to Server Payment Log Server Server 4. Digitally sign and Encrypt Response 5. Send ciphered message to client Address Change 1. Verify client 1. Log Transactions 1. Update Signature 2. Send Log to Customer 2. Decrypt IBIP Record client Infrastructure 2. Send Request Transaction 3. SendRequest Summary to to Customer Log Server Server Address Cleanse 1. Verify client 1. Log Transactions 1. Cleanse Signature 2. Send Log to TPS Addresses 2. Decrypt (IBIP 2. Create file of client Infrastructure) addresses Request 3. Notify 3. SendRequest Transaction to Customer Server Server 4. Send 4. Digitally Transaction sign and Summary to Encrypt Log Server Response 5. Send ciphered message

Detailed Description Paragraph Table (3):
TABLE III Cryptographic Keys Certificates PSD Registers Client Public Client.sub.n Indicium Client PSD 20 Authentication Key Certificate (RSA) Client Private Indicium Server Certificate Master PSD 40 Key (DSA) Server Public/Private Certification Authentication Keys Authority Certificate (RSA) Server Internal Private TPS 6 Certificates Key (DSA)

Detailed Description Paragraph Table (4):
TABLE IV Security Transaction Security Audit Control Cash Mgt. Compliance Registration SSL3.0 Creation of Customer Record CMLS License Request RC2 Store X.509, Key Pair, License ID MD5 Acquisition SSL3.0 (Download) Purchase RSA Authentication Log Purchase Request Transfer of funds to USPS Daily Log File Upload to CMRS Integrity (Hash) Log Prior/Current Register States banking authority RC2 Encryption Add to individual descending Session Key register Deduct from master descending register Refund RSA Authentication Log Refund Request Add to individual descending Daily Log File Upload to CMRS Integrity (Hash) Log Prior/Current Register States register RC2 Encryption Deduct from master Session Key descending register Printing RSA Authentication Log Indicia(um) Deduct from individual Indicia Log available Integrity (Hash) Log Prior/Current Register States descending register RC2 Encryption Session Key Cleansing RSA Authentication Log Addresses Address Log available Integrity (Hash) RC2 Encryption Session Key Address Change RSA Authentication Log Prior/Current Addresses Daily Log File Upload Integrity (Hash) to MATS/NMATS RC2 Encryption Submit update request to CMLS Session Key Software Change SSL3.0 Store prior software Postage Rate SSL3.0 version off-line Update

CLAIMS:

1. A system for conducting Internet based financial transactions comprising:

h e b b g e e e f c e c e ge

a client having a processor, a printer, a client authentication module, a module for issuing a transaction request, and a unique digital signature;

a server having a network including a transaction server, a transaction database, a server authentication module, and a receipt generation module; and

an internet connection between the client and the server network;

wherein the transaction execution system further comprises:

an authentication module, wherein the client authentication module and the server authentication modules communicate via the internet connection and are authenticated to each other;

a transaction module wherein, in response to the client and server being authenticated, the client issues a transaction request to the server and the transaction server, in response to a client transaction request, executes an electronic payment transaction at the server and records the transaction in the transaction database, and wherein the server receipt generation module, in response to an executed electronic payment, generates a receipt and transmits said receipt to the client, said receipt comprising the client digital signature and a data set uniquely identifying the executed transaction; and

wherein the receipt is printable by the client printer and the printed receipt is an evidence of payment for the executed transaction.

4. The system of claim 1 wherein the server further comprises a first server, a firewall and a single TCP/IP port, the first server is connected to the interact connection, the firewall is interposed between the server first server and transaction server so that the transaction server is connected to the network through the single TCP/IP port and the firewall comprises a communication module that operates to limit communications between the internet and the transaction server to client transaction requests identifying the single TCP/IP port.

9. The system of claim 1 wherein the client further comprises a client public key, a client private key, and a client identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the client public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the client private key and client public key based on said hash; wherein the server further comprises a server public key, a server private key, and a server identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the server public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the server private key and server public key based on said hash; and wherein said authentication module utilizes said public and private keys of said client and server to perform the authentication.

10. A method for conducting Internet based financial transactions comprising:

providing a client having a processor, a printer, a client authentication module, a module for issuing a transaction request, and a unique digital signature;

providing a server having a network including a transaction server, a transaction database, a server authentication module, and a receipt generation module;

connecting the client to the server network via an internet connection;

authenticating the client and the server to each other;

issuing a transaction request from the client to the server;

in response to a client transaction request, executing an electronic payment transaction at the transaction server and recording the transaction in the transaction database, generating a receipt at the server receipt generation module, providing said receipt with the client digital signature and a data set uniquely identifying the executed transaction, and transmitting said receipt to the client; and

printing said receipt using the client printer, wherein the printed receipt is an evidence of payment for the executed transaction.

13. The method of claim 10 wherein providing the server further comprises providing a single TCP/IP port connecting the first server to the internet connection with said client and limiting communications between the internet and the transaction server to client transaction requests identifying the single TCP/IP port.

18. The system of claim 10 wherein authenticating the client and the server further comprises providing the client with a client public key, a client private key, a client identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the client public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the client private key and client public key based on said hash; providing the server a server public key, a server private key, a server identification password, a hash module for performing a hash algorithm based on an input data, a hash of at least one of the server public key, private key and identification password, and an encryption/decryption module for encrypting and decrypting at least one of the server private key and server public key based on said hash; and using said public and private keys of said client and server to perform the authentication.